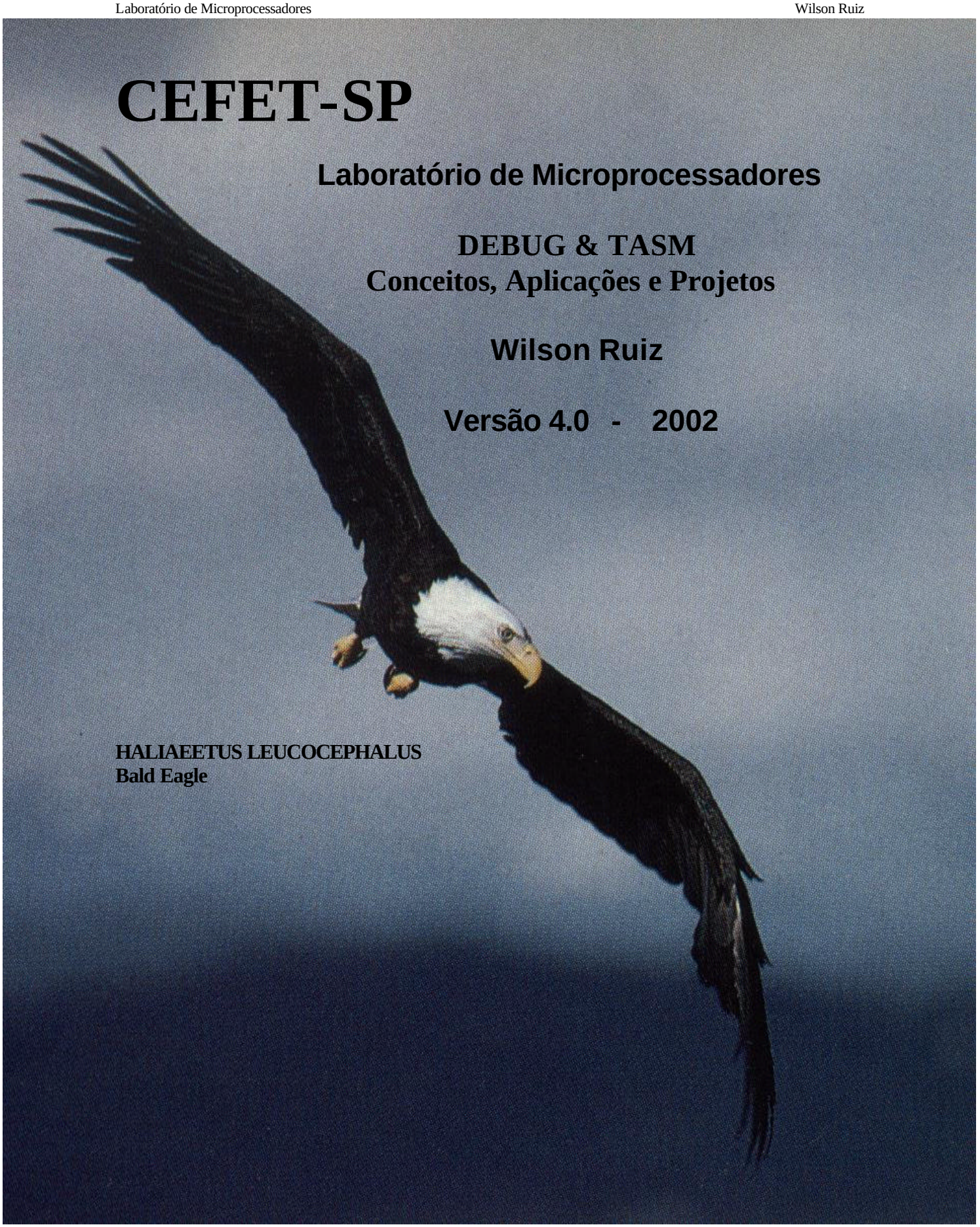
CEFET-SP

Laboratório de Microprocessadores

DEBUG & TASM
Conceitos, Aplicações e Projetos

Wilson Ruiz

Versão 4.0 - 2002



HALIAEETUS LEUCOCEPHALUS
Bald Eagle

CAPÍTULO 1 – CONCEITOS BÁSICOS

1.1 Apresentação:

Este material, oriundo do trabalho em disciplinas relacionadas ao assunto, tem o objetivo de apenas de introduzir o leitor/aluno no universo da programação assembly (dos microprocessadores das sub-famílias Intel: 80x86, 80x88 e Pentium, conhecidas por i86) e também de projetos em hardware, apresentando conceitos e sugerindo aplicações de uma forma muito direta, não tendo a pretensão de esgotar o assunto.

Razões para aprender programação assembly:

- Conhecer melhor o funcionamento do hardware (microprocessador, periféricos, memórias, etc), permitindo o desenvolvimento de projetos (de hardware e/ou software) mais eficientes.
- Obter o controle total do PC.
- Os programas em assembly são em geral mais rápidos, menores e mais poderosos (até certos limites) que os criados em outras linguagens.
- Possibilidades da inclusão de rotinas assembly em programas escritos em linguagens de níveis alto ou médio (C, Delphi etc).
- Fácil adaptação do programador a novas tecnologias como; novas famílias de microprocessadores e microcontroladodes.

1.2 Hardware básico.

Características gerais:

- 14 registradores de 16 bits.
- Capaz de endereçar diretamente 1Mbyte de memória.
- Capaz de acessar 65.535 (64Kendereços) portas de I/O

Registradores de uso geral:

AX	**BX**	**CX**	**DX**	**(16 bits)**
AH/AL	**BH/BL**	**CH/CL**	**DH/DL**	**(8 bits)**

AX: Acumulador

- Uso geral e para operações aritméticas e lógicas (registrador que é especificado pelo maior número de instruções).

BX: Base

- Uso geral e para indexar tabelas de memória (ex.: índice de vetores).

CX: Contador

- Uso geral e como contador de repetições (loop's) e movimentação repetitiva de dados.

DX: Dados

- Uso geral e para operações aritméticas.

Registradores de Segmento:

São usados para a identificação de segmentos específicos de memória.

CS: Code Segment

- Endereça o segmento de código do programa que está sendo executado.
- O valor do CS não pode ser atualizado diretamente.

DS: Data Segment

- Endereça o segmento de dados do programa.

SS: Stack Segment

- Endereça o segmento de pilha do programa.

ES: Extra Segment

- Usado para endereçar um segmento de dados adicional.

Registradores Ponteiros ou de Deslocamento

Usados em conjunto com registradores de segmento.

IP: Instruction Pointer

- Usado em conjunto com o CS para apontar o byte da próxima instrução a ser executada (endereço de off-set, dentro do segmento).
- Não há acesso direto.

SP: Stack Pointer

- Usado em conjunto com o SS para apontar o topo da pilha (stack).

BP: Base Pointer

- Usado para indicar a base da pilha.
- Particularmente importante para a passagem de parâmetros em linguagem de alto nível (como C).

SI & DI: Source Index e Destination Index

- Usados em instruções de string.
- Podem ser usados para indexar acessos a memória.

Registrador de Flags:

Consiste em um registrador de bits de controle (flag), que pode ter seus testados individualmente no programa por instruções condicionais.

Possui 16 bits dos quais apenas 9 tem função para o usuário.

OF: Overflow Flag
- Setada quando ocorre overflow aritmético.

DF: Direction Flag
- Setada para auto-incremento em instruções de string.

IF: Interruption Flag
- Permite que ocorram interrupções quando setada.
- Pode ser setada pelo sistema ou pelo usuário.

TF: Trap Flag (debug)
- Usada por debugadores para executar programas passo a passo.

SF: Signal Flag
- Resetada (SF=0) quando um resultado for um número positivo ou zero e setada (SF=1) quando um resultado for negativo.

ZF: Zero Flag
- Setada quando um resultado for igual a zero.

AF: Auxiliar Flag
- Setada quando há overflow entre o nibble menos significativo e o mais significativo de um byte.

PF: Parity Flag
- Setada quando o número de bits 1 de um resultado for par (paridade par).

CF: Carry Flag
- Setada se overflow no bit mais significativo do resultado.

Memória Segmentada:

- A memória é dividida em um número arbitrário de segmentos.
- O tamanho dos segmentos varia de 16 bytes a 64Kbytes.
- Cada segmento se inicia em um endereço sempre múltiplo de 16.
- O endereço de segmento é composto por duas words (16 bits cada).
- A anotação usada para representar um endereço é:
  Segmento:off-set

Exemplo:

Endereço lógico:

015Bh : 0009h (CS = 015Bh e IP = 0009h)
015Bh: endereço de segmento
0009h: endereço de off-set

Endereço físico:

```
  015B0h
+ 0009h
 ======
  015B9h
```

Pilha Operacional:

- Usada para o armazenamento temporário de dados.
- Serve para a passagem de parâmetros.
- É usada do endereço maior para o endereço menor.
- São enviados para a pilha e retirados desta apenas words (16 bits).
- Controle quanto a "overflow" da pilha é responsabilidade do programador.

1.3 Criação de programas.

Para a criação de grandes programas são necessários os seguintes passos:

* Desenvolvimento do algoritmo:
  Estágio em que o problema é estabelecido e uma solução escolhida, possibilitando assim a criação dos respectivos fluxogramas.

* Codificação do algoritmo:
  Consiste em escrever o programa em alguma linguagem de programação (linguagem assembly neste caso específico), tomando como base a solução adotada no passo anterior (constituindo-se o "programa fonte").

* Transformação para a linguagem de máquina (compilação):
  Criação do "programa objeto" (a partir do "programa fonte"), escrito como uma seqüência de zeros e uns que podem ser interpretados pelo microprocessador. Esta etapa é realizada integralmente pelo compilador.

* Eliminação de erros detectados no programa na fase de teste:
  A correção em geral, requer a repetição de todos os passos, com observação atenta.

1.4 Os ambientes de programação.

Para a criação de programas em assembly trabalhamos com duas opções: o DEBUG e o TASM - Turbo Assembler da Borland, abordados nos próximos capítulos.

1.5 Estrutura das instruções em Assembly.

Nas linhas do código em Linguagem Assembly existem três partes notáveis: a primeira é o nome da instrução a ser executada; a segunda, são os parâmetros do comando; a terceira, os operandos.

Exemplo 1: ADD AH, BH

Aqui "ADD" é o comando a ser executado, neste caso uma adição e, os registradores "AH" e "BH" são os parâmetros.

Exemplo 2: MOV AL, 25

Neste exemplo "MOV" é a instrução a ser executada, neste caso uma movimentação de dados, "AL" é o parâmetro e o byte 25h o operando (valor em sempre em hexadecimal no DEBUG e opcionalmente em binário, decimal ou hexadecimal no TASM).

Os nomes das instruções nesta linguagem são constituídos de 2, 3 ou 4 letras (oriundas de abreviações de frases em inglês). Estas instruções são chamadas mnemônicos (algo fácil de ser memorizado) e quando compiladas são associadas ao seu respectivo código de operação (Op. Code), representando a função que o microprocessador executará.

1.6 Observações

Este procedimento experimental é apresentado em itens numerados, cada qual abordando de forma objetiva o funcionamento de um determinado comando ou função, algumas características relevantes a este e/ou exercícios práticos e projetos sobre o mesmo.

Para um melhor desempenho do leitor/aluno durante a execução desse roteiro, é necessário que o mesmo seja seguido na ordem crescente dos itens, do respectivo capítulo, e que o conteúdo de cada tópico seja lido integralmente.

Para indicar o início das observações, ou comentários adicionais, pertinentes ao respectivo item do roteiro é usado o símbolo:

→

Evitando que o texto explicativo seja confundido com o próprio programa.

Quando as explicações forem feitas sobre um determinado comando ou registrador do microprocessador, o nome deste aparecerá sempre entre aspas “ “.

Quando as explicações forem feitas sobre uma determinada instrução em um programa apresentado, será usado um ponto-e-vígula “ ; ” para separar o texto desta instrução.

Nos exemplos, onde existirem desvios, estes podem conter o endereço do ponto de entrada no programa (quando a aplicação é feita no DEBUG) ou de outra forma, simplesmente um nome (label) que representará esse endereço (para o TASM), assim o ponto de entrada no programa será representado também por esse nome seguido de dois pontos “ : ” para separá-lo da instrução.

CAPÍTULO 2 – PRÁTICA COM O DEBUG

2.1 O DEBUG - Introdução.

Neste capítulo vamos usar o DEBUG, uma vez que este possui ótimas características didáticas e podemos encontrá-lo em qualquer PC com o MS-DOS / Windows, podendo ser encontrado normalmente no diretório C:\DOS ou C:\WINDOWS.

DEBUG é um ambiente para desenvolvimento e depuração de programas em assembly dos microprocessadores das sub-famílias Intel 80x86, 80x88 e Pentium (i86), e pode apenas criar arquivos com a extensão ".COM", que devido as suas características, não podem exceder 64 Kbytes de comprimento, e também devem sempre iniciarem no endereço de memória 0100H dentro do segmento determinado pelo sistema. É importante observar isso, pois deste modo os programas ".COM" não são relocáveis.

Usando este ambiente, entre outras coisas, é possível, visualizar os valores dos registradores internos da CPU (microprocessador) e executar o programa na forma passo-a-passo, além da compilação automática das instruções, recursos interessantes durante o início do aprendizado e para o desenvolvimento de projetos.

Inicialmente são apresentados os comandos mais importantes do DEBUG e após, gradativamente, são propostos exercícios e projetos relativos aos itens estudados.

Para a execução de todos os comandos mostrados nos exemplos e solicitados nos exercícios, é necessário acionar a tecla ENTER após a digitação da linha correspondente.

O sinal de prompt do DOS adotado nos exemplos é:

C:\>

O sinal de prompt do DEBUG é:

-

Assim os comandos a direita deste prompt, são os que devem ser digitados.

Para representar o valores numéricos apresentados na tela do microcomputador que não possuem significado (remanecentes na memória), ou podem ter outros valores em diferentes máquinas, é adotado neste roteiro o símbolo ## (para um byte) e #### (para dois bytes).

2.2 Entrar no Debug.

→ Digitar o comando:

C:\>DEBUG

2.3 Sair do Debug.

→ Digitar o comando:

- Q

2.4 Chamar o Help do Debug.

→ Digitar o comando:

- ?

→ É mostrada a seguinte tela:

help	?
assemble	A [endereço]
compare	C intervalo endereço
dump	D [intervalo]
enter	E endereço [lista]
fill	F intervalo lista
go	G [=endereço] [endereços]
hex	H valor1 valor2
input	I porta
load	L [endereço] [unidade] [primeiro-setor] [número]
move	M intervalo endereço
name	N [caminho] [lista-de-parâmetros]
output	O porta byte
proceed	P [=endereço] [número]
quit	Q
register	R [registrador]
search	S intervalo lista
trace	T [=endereço] [valor]
unassemble	U [intervalo]
write	W [endereço] [unidade] [primeiro-setor] [número]
allocate expanded memory	XA [#páginas]
deallocate expanded memory	XD [identificador]
map expanded memory pages	XM [páginaL] [páginaP] [identificador]
display expanded memory status	XS

→ Os principais comandos serão abordados no decorrer do texto.

→ Sempre consultar o "Help" para esclarecer dúvidas sobre os comandos.

2.5 Aritmética em Hexadecimal.

→ Digitar os comandos:

- H 3 2

- H 12 AA

- H 3D5C 2A10

→ O comando "H" imprime, numa linha abaixo, a soma (1º valor) e a diferença (2º valor) entre os dois números digitados.

→ Confirmar os valores apresentados e testar outros valores de até 16 bits.

→ Sua principal utilidade é calcular o tamanho de um determinado programa (em hexadecimal), necessário para a gravação de um arquivo executável.

2.6 Verificar o conteúdo de todos os registradores.

→ Digitar o comando:

- R

→ É apresentada uma listagem semelhante a esta (diferente nos segmentos):

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1669  ES=1669  SS=1669  CS=1669  IP=0100   NV UP EI PL NZ NA PO NC
1669:0100 89D8            MOV       AX, BX
-
```

→ Verificar na listagem:
Os registradores de uso geral: AX, BX, CX e DX com os seus respectivos conteúdos.
Os registradores de propósito específico: SP, BP, SI, DI, DS, ES, SS, CS, e IP com seus respectivos conteúdos.
Todos os registradores de segmento com os mesmos valores.
Indicação do status de todas as flags.
O valor de IP e o endereço mostrado na última linha.
Exemplo: Para os valores seguintes

1669:0100 1669 → início do segmento
0100 → estamos neste off-set dentro do segmento
O Debug sempre coloca "IP" em 0100h ao ser iniciado e também encarrega-se de escolher os segmentos convenientemente.
Em todas aplicações no DEBUG, trabalharemos sempre dentro de um único segmento.

A próxima instrução a ser executada, é mostrada em mnemônico e em hexadecimal na última linha.

2.7 Verificar e alterar o conteúdo de um registrador qualquer.

→ Digitar os comandos:

- R AX
AX 0000
_

→ Digite um novo valor para AX.

→ Verifique se o dado foi armazenado, digitando o comando:

- R

→ O comando "R" seguido do nome do registrador indica que queremos ver o seu conteúdo e depois altera-lo. Este comando só permite a alterações de 16 bits, não sendo possível portanto uma mudança de somente a parte alta ou baixa do respectivo registrador.

→ Testar o comando em outros registradores de uso geral.
Verificar as alterações.
Sair do Debug.
Entrar novamente no Debug.
Verificar o conteúdo dos registradores alterados anteriormente.

2.8 Examinar o conteúdo de uma posição de memória.

→ Digitar os comandos:

- E 100

- E 101

- E 102

→ O comando “E” mostra o conteúdo de um endereço especificado.

2.9 Alterar o conteúdo de uma ou várias posições de memória.

→ O comando “E” permite, além de examinar, alterar o conteúdo de um endereço na memória, bastando para isso digitar os novos bytes, separados por espaço, e pressionar o ENTER para finalizar o comando.

→ Digitar o comando:

- E 100
####:0100 ##.01 ##.D8

→ Usando o comando “E” verifique a entrada dos bytes anteriores.
01 e D8 são os OP-CODES (códigos de operação) da seguinte instrução em assembly: ADD AX,BX.

→ Usar o comando “R” para verificar a última linha da listagem.

2.10 Executar uma única instrução.

→ Com o comando “R” carregar 03A7 em “AX” e 092A em “BX”.

→ Digitar o comando:

- T

→ Verificar:
IP = 0102 ;este registrador foi deslocado e indica o endereço da próxima instrução a ser executada.
AX = 0CD1
BX = 092A
Observar a última linha da listagem

→ O comando “T” permite a execução de uma instrução de cada vez (execução passo-a-passo) além de apresentar a listagem dos registradores.

2.11 Exercício.

→ Executar novamente a mesma instrução com os resultados do item “2.10”; para isso é necessário usar o comando “R” para posicionar “IP” em 0100 e, só então, usar o comando “T”.

→Sempre devemos checar o valor atual de "IP" e a instrução mostrada na última linha da listagem de registradores, antes de executar o comando "T".

2.12 Exercício.

→ Digitar o comando:

- E 0100
: 0100 ## . 29 ## . D8

→ Alterar o valor de "IP" para 0100
Verificar o mnemônico da instrução digitada.
Carregar valores de teste nos registradores envolvidos.
Executar a instrução
Verificar os resultado

2.13 Entrada de instruções em mnemônico.

→ Digitar o comando e o fragmento de programa a seguir:

```
- A 0100
#### : 0100  MOV     AX, 0123
#### : 0103  MOV     BX, 4567
#### : 0106  MOV     CX, 89AB
#### : 0109  ADD     AX, BX
#### : 010B  MOV     DX, AX
#### : 010D  MOV     AX, BX
#### : 010F  XOR     AX, CX
```

→ O comando "A" possibilita a digitação direta de instruções em mnemônicos a partir do endereço indicado por "IP" (p/ apenas "A") ou a partir de um endereço especificado (p/ "A 0100", por exemplo)

→ As instruções podem ser digitadas em letras maiúsculas ou minúsculas indiferentemente.

→ Os espaços existentes nas instruções não são importantes para o DEBUG, porém é útil ter o hábito de digita-los pois os compiladores são mais exigentes com relação a sintaxe das instruções.

→ Lembrar sempre que todos os valores numéricos digitados são assumidos pelo DEBUG, como expressos em hexadecimal.

→ Quando uma instrução é digitada de forma errada, o DEBUG mostra uma mensagem de erro e possibilita a sua imediata correção.

→ Estabelecer "IP=0100" e executar o programa anterior passo-a-passo, até a instrução XOR AX, CX (NUNCA execute instruções ou trechos desconhecidos de programas, pois o microcomputador poderá "travar"), analisar o resultado mostrado na listagem dos registradores, após cada instrução.

2.14 Listagem do programa digitado.

→ Digitar o comando:

- U 0100

→ Identificar na tela apresentada o programa anteriormente digitado

→ O comando "U" permite verificar a listagem do programa na memória a partir do endereço de "IP" (p/ apenas "U") ou a partir de um endereço especificado (p/ "U 0100", por exemplo).

→ Para sucessivos comandos "U", o conteúdo da memória é mostrado seqüencialmente.

2.15 Examinar o conteúdo de uma região de memória.

→ Digitar o comando:

- D

→ Verificar que o conteúdo numérico de 128 posições consecutivas de memória é apresentado na tela em linhas de 16 bytes cada, com uma faixa de endereços escolhida dentro do segmento, a partir do valor atual de "IP". A direita da listagem temos os caracteres ASCII correspondentes (quando um ponto "." é apresentado, este indica que o caractere é realmente um ponto ou é um caractere de controle).

→ Para sucessivos comandos "D" a memória pode ser rapidamente vasculhada.

→ Para verificar o conteúdo de uma região não iniciada no valor atual de "IP" (por exemplo :0200h) digitaríamos:

- D 0200

→ Para verificar o conteúdo de apenas uma pequena região de memória (por exemplo de :0150h a :0158h) digitaríamos:

- D 0150 0158

→ Testar as possibilidades desse comando.

2.16 Preenchimento de regiões de memória rapidamente.

→ Digitar o comando:

- F 0150 016F 22

→ Com o comando "D" verificar o conteúdo da memória do endereço :0150h ao endereço :016Fh.

→ O comando "F" preenche o trecho de memória determinado com o byte fornecido (no exemplo 22h).

→ Digitar o comando:

- F 0170 018F 00 11 22 33 44

→ Com o comando "D" verificar o conteúdo da memória do endereço :0170h ao endereço :018Fh.

→ Este comando preenche a região de memória especificada com a seqüência de bytes fornecida (no exemplo 00h 11h 22h 33h 44h).

2.17 Copiar um bloco contínuo de memória.

→ Usando o comando "A 0100", digite o seguinte fragmento de programa:

```
:0100  MOV  AL, 00
       MOV  AH, 01
       MOV  DL, 06
       MOV  DH, 07
```

→ Supondo que por um erro, você esqueceu de digitar entre a 2ª e 3ª instruções estas outras:

```
       MOV  BL, 02
       MOV  BH, 03
       MOV  CL, 04
       MOV  CH, 05
```

→ Como o grupo de instruções a ser inserido tem um total de 8 bytes, copiar as duas últimas instruções do programa digitado para uma região 8 bytes abaixo, digitando o seguinte comando:

- M 0104 0107 010C

→ Verificar com o comando "U".

→ Copia os dados da faixa de endereços de :0104h a :0107h para a partir do endereço :010Ch.

→ Com o comando "A 0104" digitar as instruções anteriormente esquecidas.

→ Verificar o final com o comando "U".

2.18 Comparar duas regiões de memória.

→ Digitar o comando:

- C 0100 010F 0150

→ Verificar os valores apresentados na tela.

→ O comando compara entre si os dados das duas regiões de memória (de :0100h a :0150h com outra que inicia-se em :0150h), mostrando diferenças.

→ Testar este comando também para regiões de memória com conteúdos idênticos.

2.19 Procurar dados na memória.

→ Digitar o comando:

- S 0100 0200 FF

→ Verificar os valores apresentados na tela.

→ O comando procura o byte FFh na região de memória especificada (no caso de :0100h a :0200h), indicando o endereço do byte encontrado.

→ Para procurar uma seqüência de bytes (por exemplo CCh DDh EEh) na mesma região de memória o comando seria:

- S 0100 0200 CC DD EE

2.20 Interrupções – Visão geral

Interrupções são estruturas usadas para transferir o controle (registradors CS:IP) de uma localização na memória para outra. Quando uma interrupção ocorre, a CPU suspende temporariamente a função atual (após terminar a instrução que está sendo executada) e executa uma rotina especial conhecida como "rotina de serviço de interrupção" (ISR). Quando a ISR termina, a CPU volta à execução no ponto imediatamente após a instrução que foi interrompida.

Existem três tipos diferentes de interrupções no PC:

- **Interrupções de Software:** usadas pelos programas para chamar rotinas na BIOS e no núcleo do DOS;
- **Interrupções de Hardware:** usadas pelos dispositivos de hardware para avisar a CPU de que estes solicitam uma função;
- **Exceções:** usadas pela própria CPU para indicar erros.

Quando ocorre uma interrupção, uma seqüência de eventos (principalmente envolvendo a preservação do status atual na pilha operacional) é executada para garantir que a CPU responda à interrupção e, mais tarde, retorne à execução do programa principal, como se nada tivesse acontecido.

Toda interrupção tem um número associado, variando de 0 a 255 (em decimal) ou de 00h a FFh (em hexadecimal). Quando uma interrupção ocorre, este número serve como um índice para a tabela conhecida como *Vetor de Interrupções*, armazenada na memória baixa. Esta tabela contém os endereços de até 256 ISR diferentes. A interrupção 05h, por exemplo, resulta na execução da ISR cujo endereço é o sexto no Vetor (interrupção 0 é o primeiro). A tabela está sempre armazenada nos primeiros bytes da memória (0000:0000).

Uma interrupção de software é gerada quando um programa executa uma instrução INT (algumas dessas serão utilizadas nesse material e outras constam na bibliografia listada).

Existem dois tipos de interrupções de hardware: **Mascaráveis** e **Não mascaráveis**.

- As interrupções não mascaráveis são executadas mesmo que o flag de interrupção esteja desativado.
- As interrupções mascaráveis são desativadas quando o flag estiver zerado.

Um dispositivo gera uma interrupção não mascarável ativando o pino NMI (Non-Maskarable Interrupt) na CPU.

Por exemplo: uma "INT 02h" (Non-Maskable Interrupt) é gerada se uma condição não mascarável, como um erro de paridade na memória, ocorrer. Esta é a única interrupção que não pode ser mascarada pela instrução CLI (Clear Interrupts).

As interrupções mascaráveis ocorrem quando um dispositivo (como o teclado, o Clock ou a porta serial) ativa uma linha de interrupção. Estes dispositivos não estão conectados diretamente na CPU, mas sim ao controlador de interrupções, cujo trabalho é acumular as requisições de interrupção dos dispositivos de hardware, priorizá-las e passá-las para a CPU uma por vez. A saída do controlador de interrupções (por exemplo o Intel 8259) é conectada a uma linha de interrupção da CPU.

Um exemplo clássico de interrupção de hardware é o que acontece quando uma tecla é pressionada no teclado. Através do controlador de interrupções, uma interrupção 09h é gerada, o que executa uma ISR do ROM BIOS. A ISR lê o código de varredura da porta do teclado, converte este código em seu equivalente ASCII e o coloca no Buffer do teclado.

O último tipo de interrupção, a exceção, é gerada pela CPU quando esta detecta uma condição de erro durante ou após a execução de uma instrução. Por exemplo, se uma instrução de divisão foi executada para o divisor igual a zero, a CPU executa uma interrupção 0.

2.21 INT 20 - Uma boa saída.

→ A interrupção INT 20 (o número 20 é expresso em hexadecimal) está presente no DOS (e portanto no Windows) e tem como função encerrar um programa em assembly sem parar o microprocessador devolvendo assim, o controle do sistema para o DOS ou para o DEBUG, dependendo do ponto, a partir do qual, o programa foi executado.

→ Digitar o comando abaixo

- A 0111
: 0111 INT 20

→ Usar o comando "U" para verificar na listagem a entrada dessa nova instrução no final do programa anteriormente digitado.

2.22 Execução integral de um programa.

→ Estabelecer "IP = 0100"

→ Usar o comando "R" para carregar 0000h nos registradores de uso geral

→ Digitar o comando seguinte:

- G

→ Uma mensagem semelhante a: "Programa finalizado normalmente" será exibida na tela devido a execução da instrução INT 20.

→ Verificar o resultado do programa com o comando "R".

→ O comando "G" permite a execução completa de um programa (partindo do endereço indicado por "IP", até encontrar o "INT 20").

→ NUNCA executar o "INT 20" como comando "T".

2.23 Execução parcial de um programa.

→ O comando "G" permite o estabelecimento de um "break point" em um programa a ser testado isto é; a execução deste até um determinado ponto para verificação de alguma determinada característica ou condição. Este recurso pode ser muito útil durante a verificação do funcionamento de um determinado trecho de um grande programa. Para testa-lo estabelecer "IP=0100" e digitar o comando abaixo:

- G 0111

→ Assim o programa é executado até o byte imediatamente anterior ao endereço indicado (endereços em hexadecimal), no caso parando o processamento antes da instrução "INT 20".

→ Agora digitar:

- G

→ Lembre-se de verificar sempre o conteúdo de "IP" antes de executar um programa.

2.24 INT 21 - Uma versátil ferramenta.

→ Assim como o "INT 20" existe um grande número de interrupções disponíveis sendo, inegavelmente, a mais versátil de todas a "INT 21".

→ Esta interrupção possui vários recursos (conhecidos como funções) que são escolhidos carregando-se inicialmente o registrador de entrada com um valor específico (inclusive um para o término de programas ".exe" que é a 4Ch).

→ A tabela seguinte lista os mais importantes desses recursos (não todos), bem como os valores necessários para a entrada e a saída correspondentes.

→ Ler com atenção as principais funções da "INT 21" relacionadas na tabela:

Função	Descrição	Entrada	Retorna	OBS
01h	Receber caractere do teclado (em ASCII) e o enviar para a tela.	AH=01h	AL=código ASCII do caractere	
02h	Enviar caractere para a tela.	AH=02h DL=código ASCII do caractere	Nada	
03h	Receber caractere da porta serial (COM1).	AH=03h	AL=código ASCII do caractere	
04h	Enviar caractere para COM1.	AH=04h DL=código ASCII do caractere	Nada	
05h	Enviar caractere para a impressora (LPT1).	AH=05h DL=código ASCII do caractere	Nada	
06h	Receber / enviar caractere.	Para receber caractere AH=06h DL=FFh Para enviar caracter AH=06h DL=código ASCII do caractere (00h-FEh)	AL=código ASCII do caractere	Não aguarda o recebimento do caractere Somente retorna o caractere se ele estiver pronto no buffer de teclado.Não reconhece CTRL+C
07h	Receber caractere do teclado sem envia-lo para a tela.	AH=07h	AL=código ASCII do caractere	Aguarda o recebimento de um caractere. Não reconhece CTRL+C
08h	Receber caractere do teclado sem envia-lo para a tela.	AH=08h	AL=código ASCII do caractere	Reconhece a combinação das teclas CRTL+C

Função	Descrição	Entrada	Retorna	OBS:
09h	Enviar string para a tela.	AH=09h DS:DX=aponta p/ uma string com 24h ($) no final.	Nada	
0Ah	Receber string do teclado.	AH=0Ah DS:DX buffer p/ armazenamento de string Composição do buffer: Byte 0: indica o número máximo de bytes possível na entrada de dados. Byte 1: será atualizado pelo DOS com o número de caracteres digitados. O fim da string é marcado com o caractere 0Dh, que embora conste da string, não é contado no byte 1.	Buffer atualizado com a string digitada pelo usuário.	Espera o usuário teclar ENTER ou entrar com um valor 0Dh. Fornece ao usuário todos os comando comuns do DOS para a edição de texto.
0Bh	Verificar estado de pressionamento de tecla	AH=0Bh	AL=FFh caso exista caractere no buffer AL=00h caso não exista caractere	
0Ch	Limpar buffer do teclado e receber caractere	AH=0Ch AL=nº da função p/ recebimento do caractere. Valores p/ AL: 01h, 06h ,07h e 08h: indicam a função desta interrupção que será chamada após o esvaziamento do buffer.	Veja descrição da função que foi escolhida em AL.	

Função	Descrição	Entrada	Retorna	OBS
0Eh	Unidades de disco presente.	AH=0Eh DL=número da unidade de disco: 00h=A, 01h=B	AL=número de unidades de disco do sistema	
19h	Obter unidade de disco ativa.	AH=19h	AL=unidade de disco ativa (00h=A, 01h=B ,...).	
2Ah	Obter data	AH=2Ah	AL=dia da semana (de 00h=domingo ... a 06h=sábado) CX=ano (de 1980 a 2099) DH=mês (de 01h=janeiro a 0Ch=dezembro) DL=dia do mês	Todos os valores retornados por esta função estarão em hexa.
2Bh	Definir data	AH=2Bh CX=ano DH=mês DL=dia do mês	AL=00h, data OK AL=FFh, data inválida	
2Ch	Obter horário	AH=2Ch	CH=hora (formato 24 horas,de 00 a 23) CL=minutos (de 00 a 59) DH=segundos (de 00 a 59) DL=centésimos de segundo (de 00 a 99)	Todos os valores retornados por esta função estarão em Hexa. Com precisão limitada ao período do clock do computador
2Dh	Definir horário	AH=2Dh CH=hora CL=minutos DH=segundos DL=centésimos de segundo	AL=00h, horário OK AL=FFh, horário inválido	Veja obs. anterior
30h	Obter versão do DOS	AH=30h	P/ versões anteriores a 5 - AL=número principal da versão e AH=secundário	
4Ch	Encerrar programa	AH=4Ch	Nada	

Hexa	Car	hexa	Car	Hexa	Car	hexa Car	hexa	car	hexa	Car
00		2B	+	56	V	81	AC		D7	
01		2C	,	57	W	82	AD		D8	
02		2D	-	58	X	83	AE		D9	
03		2E	.	59	Y	84	AF		DA	
04		2F	/	5A	Z	85	B0		DB	
05		30	0	5B	[	86	B1		DC	
06		31	1	5C	\	87	B2		DD	
07		32	2	5D	]	88	B3		DE	
08		33	3	5E	^	89	B4		DF	
09		34	4	5F	_	8A	B5		E0	
0A		35	5	60	`	8B	B6		E1	
0B		36	6	61	a	8C	B7		E2	
0C		37	7	62	b	8D	B8		E3	
0D		38	8	63	c	8E	B9		E4	
0E		39	9	64	d	8F	BA		E5	
0F		3A	:	65	e	90	BB		E6	
10		3B	;	66	f	91	BC		E7	
11		3C	<	67	g	92	BD		E8	
12		3D	=	68	h	93	BE		E9	
13		3E	>	69	i	94	BF		EA	
14		3F	?	6A	j	95	C0		EB	
15		40	@	6B	k	96	C1		EC	
16		41	A	6C	l	97	C2		ED	
17		42	B	6D	m	98	C3		EE	
18		43	C	6E	n	99	C4		EF	
19		44	D	6F	o	9A	C5		F0	
1A		45	E	70	p	9B	C6		F1	
1B		46	F	71	q	9C	C7		F2	
1C		47	G	72	r	9D	C8		F3	
1D		48	H	73	s	9E	C9		F4	
1E		49	I	74	t	9F	CA		F5	
1F		4A	J	75	u	A0	CB		F6	
20	Space	4B	K	76	v	A1	CC		F7	
21	!	4C	L	77	w	A2	CD		F8	
22	“	4D	M	78	x	A3	CE		F9	
23	#	4E	N	79	y	A4	CF		FA	
24	$	4F	O	7A	z	A5	D0		FB	
25	%	50	P	7B	{	A6	D1		FC	
26	&	51	Q	7C	|	A7	D2		FD	
27	‘	52	R	7D	}	A8	D3		FE	
28	(	53	S	7E	~	A9	D4		FF	
29	)	54	T	7F		AA	D5			
2A	*	55	U	80	Ç	AB	D6			

2.25 Códigos de caracteres ASCII.

→ Analisar rapidamente o conteúdo da tabela anterior.

→ O padrão ASCII (American Standard Code for Information Interchange) original é constituído por 128 caracteres, com códigos decimais de 0 a 127 (de 00h a 7Fh), tornando-se um código universal, usado em diferentes famílias de computadores, porém, o conjunto apresentado na tabela tem o dobro de códigos, incluindo-se ali os códigos de 128 até 255 (de 80h a FFh), seqüência de códigos padronizada apenas para a família PC, más eventualmente adotada de forma parcial para equipamentos de outras famílias. Em geral os códigos que compõem a metade adicional da tabela, são chamados caracteres ASCII estendidos, usados como caracteres de controle e não reproduzidos na tabela anterior.

2.26 Envio de caracter para o monitor de vídeo.

→ Digitar o programa abaixo e executar a seqüência de comandos:

- A 0100

```
MOV  AH, 02
MOV  DL, 2A
INT  21
INT  20
```

- U 0100

- R

- G

→ Analisar o resultado

2.27 Exercício: Imprimir outras letras.

→ Consultando a tabela ASCII fornecida, testar o programa anterior com outros códigos (imprimindo outros caracteres).

2.28 Nomear um programa para futura gravação em disco.

→ Para dar nome ao programa digitar o comando:

- N letra.com ; se desejado usar outro nome seguido de ".COM"

→ Agora o programa anterior possui o nome digitado (com no máximo 8 caracteres, no padrão DOS; e extensão .com)

2.29 Salvar o programa em disco.

→ Para gravar o programa (arquivo executável) é necessário indicarmos o tamanho do mesmo para o Debug assim, utilizaremos o comando "H" para calcular o número de bytes do mesmo da seguinte forma:

- H 108 100

→ 0100h = endereço de início do programa
0108h = o byte seguinte ao término da última instrução do programa
0208 0008 = respectivamente a soma e a diferença dos valores digitados

→ O uso do comando "H" para esse cálculo pode ser muito útil na determinação do comprimento de programas maiores.

→ O Debug usa o par de registradores "BX:CX" para o tamanho do programa a ser gravado portanto, usando o comando "R" definir "BX=0000" e "CX=0008".

→ Digitar o comando:

- W

→ Ler a mensagem na tela.

→ Sair do Debug (comando "Q").

→ Usar o comando "DIR" do DOS para verificar o arquivo criado

C:\>DIR LETRA.COM

→ Executar este programa diretamente no DOS simplesmente digitando o seu nome:

C:\>LETRA

→ Analisar o resultado

→ Para gravarmos este programa em um disquete teríamos a seguinte seqüência de comandos:

- N A:LETRA.COM

- W

2.30 Leitura de um arquivo.

→ Para ler (carregar) o arquivo do disco para o DEBUG, e verificar o seu conteúdo, executar a seguinte seqüência de comandos:

- N LETRA.COM
- L
- U 0100

→ Ou para o arquivo em disquete teríamos:

- N A:LETRA.COM
- L
- U 0100

2.31 Execução passo-a-passo de um programa com interrupções do DOS.

→ Para ser possível a execução de um programa na forma passo-a-passo e conseqüentemente a sua análise NÃO devemos usar o comando "T" para executar qualquer interrupção, pois estas na verdade, são longos programas e não simples instruções e assim, o fluxo do processamento seria desviado, haveria uma mudança de segmento de memória e provavelmente ficaríamos perdidos.

→ É possível contornar-se esse problema tomando-se o seguinte cuidado:
Executar novamente o programa anterior passo-a-passo usando o comando "T" até o endereço onde a próxima instrução a ser executada é a "INT 21".
Usar agora o novo comando "P" para executar a interrupção indicada.

→ Outra opção seria usar o comando "G" como um break pointer até imediatamente após a interrupção em questão.

2.32 Impressão de string na tela.

→ Digitar o programa:

```
MOV  AH, 09
MOV  DX, 0200
INT  21
INT  20
```

→ Com o comando “E” entrar a seguinte seqüência de bytes:

- E 0200
47, 72, 61, 6E, 64, 65, 20, 50, 72, 6F, 66, 2E, 20, 57, 69, 6C, 73, 6F, 6E, 24

→ Executar o programa integralmente (com o comando “G”)

→ Esta função do “INT 21” imprime um string inteiro (endereçado por “DX”) e finalizado pelo caracter “$” (24h no código ASCII).

2.33 Entrada da string diretamente:

→ Digitar a seqüência de comandos:

- A 0150

:0150 DB “Isto é um teste$”

- D 0150

→ Com o comando “D” é possível verificar que a string anteriormente digitada é mostrada no lado direto da tela.

→ O sub-comando “DB” possibilita a entrada direta de strings na memória (entre aspas duplas ou simples), para depois serem chamadas pela interrupção correspondente.

2.34 Exercício.

→ Usar a tabela de códigos ASCII fornecida e modificar o programa anterior para imprimir strings diferentes.

→ Usar o comando “D” para verificar a string na memória.

2.35 Leitura de caractere do teclado.

→ Digitar o programa a partir de :0100

```
MOV  AH, 01
MOV  AL, FF
INT  21
INT  20
```

→ Executar estabelecendo um "break pointer" na última instrução isto é, com o comando: G 0106, que indica o endereço posterior ao término da execução do trecho de programa escolhido.

→ Você verá um cursor piscando e o DOS esperando que uma tecla seja acionada.

→ Pressionar uma tecla (por exemplo "a") e depois verificar o conteúdo de "AL" na listagem apresentada ou pelo comando "R".

→ Comparar o valor encontrado em "AL" com o da tabela ASCII.

→ O DOS carrega o código ASCII do caractere da tecla pressionada no registrador "AL" (notar que o conteúdo de "AL", que era inicialmente FFh, foi alterado). O "break pointer foi necessário pois o "INT 20" altera o conteúdo dos registradores.

→ Para sair do "break pointer" basta apenas digitar o comando "G".

2.36 Exercício.

→ Repetir o procedimento anterior para outras teclas (letras minúsculas, maiúsculas, números e símbolos).

2.37 Leitura de teclas especiais.

→ Repetir o procedimento anterior para as teclas especiais, isto é, com o código ASCII estendido (por exemplo a tecla F1).

→Para cada tecla especial o DOS envia dois caracteres, um após o outro. O primeiro é sempre "00h", indicando que o próximo caractere é o código de varredura de uma tecla especial, assim você verá "00h" no registrador "AL" e o caractere ASCII equivalente ao segundo byte impresso na tela.

2.38 Exercício.

→ Repetir o procedimento anterior para outras teclas especiais.

2.39 Obter a data.

→ Digitar o programa a partir de :0100h

```
MOV  AH, 2A
INT  21              ;continua...
```

```
INT   20
```

→ Executar o programa estabelecendo um "break pointer" na última instrução.

→ Na listagem apresentada ou com o comando "R" verificar o conteúdo dos registradores "AL, CX, DH e DL".

→ Comparar os valores encontrados com os esperados (de acordo com as funções da "INT 21", constantes na tabela fornecida).

2.40 Definir data.

→ Usar a "INT 21" para definir uma nova data para o sistema (consultar a tabela das funções da "INT 21").

→ Ler essa nova data.

→ Voltar para a data atual.

2.41 Obter o horário.

→ Digitar o programa abaixo:

```
MOV  AH, 2C
INT  21
INT  20
```

→ Executar o programa estabelecendo um "break pointer" na última instrução.

→ Na listagem apresentada ou com o comando "R" verificar o conteúdo dos registradores "CH, CL, DH e DL".

→ Comparar os valores encontrados com os esperados (de acordo com as funções da "INT 21", constantes na tabela fornecida).

→ Posicionar "IP" e executar novamente o programa; e comparar os novos valores com os anteriores.

2.42 Definir horário.

→ Usar a "INT 21" para definir um novo horário para o sistema (consultar a tabela das funções da "INT 21").

→ Ler esse novo horário.

→ Voltar para o horário atual.

→ O horário é calculado pela contagem dos "pulsos de clock" da máquina, assim nenhum evento pode ser temporizado com precisão acima da taxa de contagem desse relógio. A freqüência desse timer é de aproximadamente 18,2 Hz; conseqüentemente a precisão será de aproximadamente 5,4 centésimos de segundo.

2.43 Informar o drive corrente.

→ Digitar o programa:

```
MOV   AH, 19
INT   21
INT   20
```

→ Executar programa estabelecendo um "break pointer" na última instrução e verificar o conteúdo de "AL" consultando a tabela fornecida.

2.44 Testes com as flags.

→ Os sinalizadores (flags) são mostrados a direita, na segunda linha, da listagem dos registradores, no DEBUG.

→ Usar o comando "R" para verificar o status atual das flags.

→ O nome e a forma de indicação de status das flags é mostrado na tabela abaixo:

Flag	nome	Quando setada	Quando resetada
OF	Overflow	OV	NV
DF	Direção	DN	UP
IF	Interrupção	EI	DI
SF	Sinal	NG	PL
ZF	Zero	ZR	NZ
AF	Carry auxiliar	AC	NA
PF	Paridade	PE	PO
CF	Carry	CY	NC

→ As principais flags são:

CARRY FLAG	CY	p/ =1	verdadeiro
	NC	p/=0	falso

ZERO FLAG	ZR	p/=1	verdadeiro
	NZ	p/=0	falso
SIGNAL FLAG	NG	p/=1	negativo
	PL	p/=0	positivo

→ Digitar o programa para testar a CARRY FLAG:

```
: 0100   MOV  AX, FFFE
         MOV  BX, 0001
         ADD  AX, BX
         INT  20
```

→ Executar o programa estabelecendo um "break pointer" na última instrução (G0108) e após, verificar o status da CARRY FLAG.

→ Substituir o valor "0001" por "0002", repetir a execução e comparar o status atual com o anterior.

→ Digitar o programa para testar a ZERO FLAG:

```
: 0100   MOV  AX, 0FFE
         MOV  BX, 0001
         SUB  AX, BX
         INT  20
```

→ Executar o programa estabelecendo um "break pointer" na última instrução (G0108) e após verificar o status da ZERO FLAG.

→ Substituir o valor "0001" por "0FFE", repetir a execução e comparar o status atual com o anterior.

→ Digitar o programa para testar a SIGNAL FLAG:

```
: 0100   MOV  AX, FFFE
         MOV  BX, 0002
         SUB  AX, BX
         INT  20
```

→ Executa o programa estabelecendo um "break pointer" na última instrução (G0108) e após verificar o status da SIGNAL FLAG.

→ Substituir o valor "FFFE" por "0001", repetir a execução e comparar o status atual com o anterior.

2.45 Exercício.

→ Usando o comando "R" carregue um valor diferente de 00h em "BL" isto é, estabeleça "BX = 00## ".

→ Digitar e executar o programa:

```
: 0100    MOV  AH, 02
          MOV  CX, 0008
          MOV  DL, 00
          RCL  BL, 1
          ADC  DL, 30
          INT  21
          LOOP 105
          INT  20
```

→ Analisar o resultado e executar outras vezes o programa para diferentes valores em "BX" (mostra na tela o conteúdo de "BL" em binário).

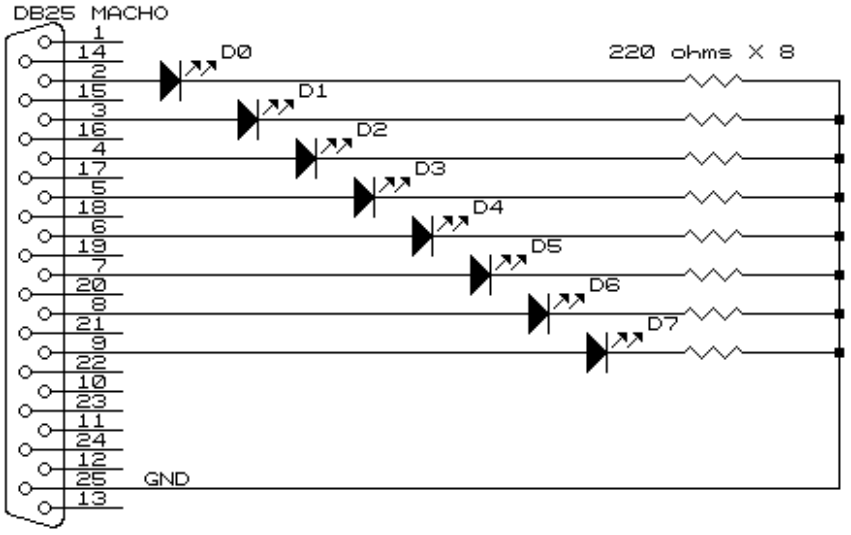
2.46 Exercício

→ Modificar o programa anterior para o mesmo imprimir um caractere "b" (significando binário) no final do número.

2.47 Porta Paralela

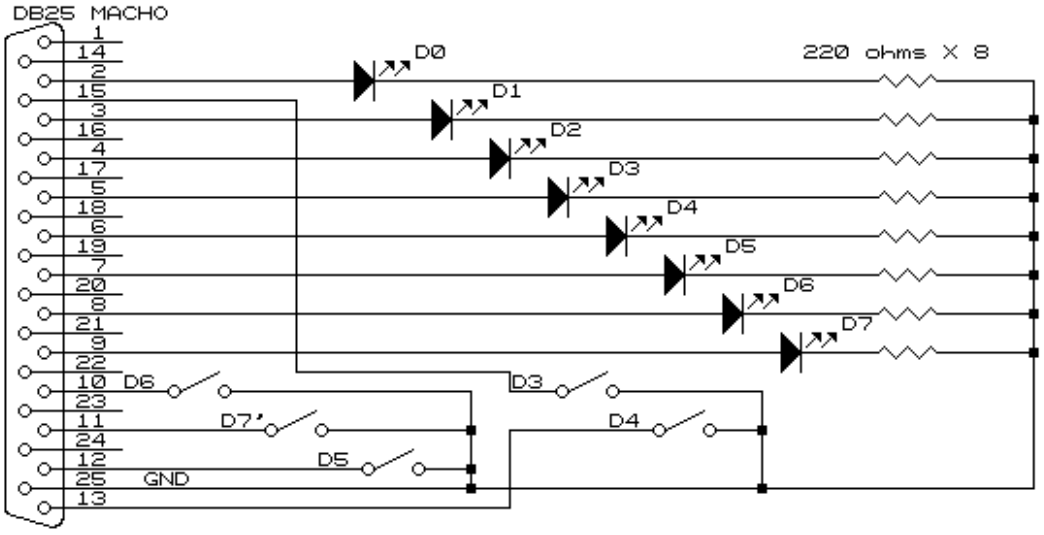
→ Analisar a tabela seguinte que mostra um resumo das características da porta paralela do PC.

Porta	BIT	Direção	Pinos (DB25)	Nome do sinal
ENDEREÇO 0378h	7	OUT	9	D7
	6	OUT	8	D6
	5	OUT	7	D5
	4	OUT	6	D4
	3	OUT	5	D3
	2	OUT	4	D2
	1	OUT	3	D1
	0	OUT	2	D0
ENDEREÇO 0379h	7	IN	11 (bit invertido)	BUSY
	6	IN	10	ACKNLG
	5	IN	12	PAPER OUT
	4	IN	13	SELECT
	3	IN	15	ERROR
ENDEREÇO 037Ah	4	---	Hab IRQ 7	---
	3	IN/OUT	17 (bit invertido)	SLCT IN
	2	IN/OUT	16	INIT
	1	IN/OUT	14 (bit invertido)	AUTO FEED
	0	IN/OUT	1 (bit invertido)	STROBE



→ Esquema eletrônico da placa de LED's.

→ Analisar o circuito utilizado para testes de saída no laboratório.



→ Sugestão de uma placa para testes de entrada e saída da porta paralela (caso haja interesse do leitor/aluno na sua montagem).

2.48 Saída de dados com comando do DEBUG.

→ Porta paralela – saída, endereço 0378h (vide tabela anterior).

→ Conectar a placa de LED's na porta paralela e digitar os comandos:

- O 0378 FF
- O 0378 00
- O 0378 55
- O 0378 AA

→ Analisar a saída de cada um desses comandos (status dos LED's).

2.49 Entrada de dados com o comando do DEBUG.

→ Porta paralela – entrada, endereço 0379h (vide tabela anterior).

→ Digitar o comando:

- I 0379

##

→ Analisar a resposta (lembrar que o "bit D7" é lido invertido pelo sistema, os bits D2, D1 e D0 não estão disponíveis sendo sempre iguais a "1" e para pinos da entrada em aberto, temos atribuído o nível lógico "1").

2.50 Saída de dados em assembly.

→ Digitar e executar o programa estabelecendo um "break pointer" na última instrução:

```
MOV  AL, 0F
MOV  DX, 0378
OUT  DX, AL
INT  20
```

→ Avaliar a resposta (status dos LED's).

→ Repetir a execução do programa anterior para outros bytes de saída (sem a necessidade do "break pointer").

2.51 Entrada de dados em assembly.

→ Usar o comando "R" para estabelecer "AX=0000".

→ Digitar e executar o programa estabelecendo um "break" pointer na última instrução:

```
MOV  DX, 0379
IN   AL, DX
INT  20
```

→ Avaliar a resposta (conteúdo de AL).

2.52 Sub-rotinas

→ Digitar o programa:

```
: 0100   MOV  DL, 41
         MOV  CX, 000A
         CALL 0200
         LOOP 0105
         INT  20

: 0200   MOV  AH, 02        ;continua...
```

```
        INT   21
        INC   DL
        RET
```

→ Executar e analisar o seu funcionamento.

2.53 Exercício

→ Modificar o programa anterior para o mesmo a apresentar na tela a lista de todos os 255 caracteres ASCII.

2.54 Sub-rotina clássica usada para temporização (pausa ou delay).

→ Analisar o funcionamento do exemplo de sub-rotina a seguir:

```
        PUSH  CX
        PUSH  DX
        MOV   CX, 00FF
L2:     MOV   DX, 00FF
L1:     DEC   DX
        JNZ   L1
        DEC   CX
        JNZ   L2
        POP   DX
        POP   CX
        RET
```

→ Esta rotina fornece uma temporização proporcional a multiplicação dos dois "loops".

→ Sempre é conveniente salvar os valores anteriores dos registradores na pilha operacional, no início de uma sub-rotina e recupera-los no final, antes da instrução RET; isso torna a sub-rotina "transparente para o programa principal", evitando a superposição de dados. A exceção a esta regra está em determinadas aplicações onde a sub-rotina deve fornecer algum resultado para o programa principal e este é armazenado em um registrador.

→ O valor do "delay" deve se ajustado (experimentalmente) para cada caso, pois além das características da aplicação, existe diferenças no "clock" de cada microcomputador, assim é de boa prática, iniciar os testes com valores pequenos para os registradores que controlam os "loops" pois, se houver exagero, a máquina poderá ficar presa em um "loop" de programa durante muito tempo.

2.55 Exemplo de um programa com temporização.

→ Digitar, executar e analisar o funcionamento do programa:

```
: 0100   MOV  DX, 0378
         MOV  AX, 00FF
         OUT  DX, AL
         CALL 0200
         MOV  AX, 0000
         OUT  DX, AL
         CALL 0200
         INT  20

: 0200   PUSH CX
         PUSH DX
         MOV  CX, 0FFF     ; se necessário ajustar o delay para o clock
         MOV  DX, FFFF     ; do microcomputador do laboratório
         DEC  DX           ; trocando os valores carregados nos
         JNZ  0208         ; registradores "CX e DX"
         DEC  CX
         JNZ  0205
         POP  DX
         POP  CX
         RET
```

2.56 "Loop" infinito

→ Se substituirmos o "INT 20" do programa principal, do item anterior, pela instrução "JMP 0100", teríamos como resultado os LED's piscando seguidamente, porém este programa assim entraria em um "loop" infinito ou seja, só sai com o RESET da máquina. Durante o projeto e testes de programas devemos tomar cuidado com essa perigosa situação pois, ela pode levar-nos a perder um bom tempo de trabalho.

→ Para evitar o "loop" infinito e poder ver os LED's da placa piscando, altere o programa anterior (bloco principal) introduzindo um contador decrescente de operações, que terminará este programa quando seu conteúdo for igual a zero (cuidado com o valor inicial desse contador para não perder um longo tempo aguardando o término do programa).

→ Nomear e gravar este programa.

2.57 Exemplo de um programa de controle com temporização.

→ Digitar, executar e analisar o funcionamento do programa:

```
: 0100   MOV  DX, 0378
         MOV  BL, 0F        ; valor inicial do contador ajustado para
         MOV  AX, 00FF      ; 15 piscadas
         OUT  DX, AL
         CALL 0200
         MOV  AX, 0000
         OUT  DX, AL
         CALL 0200
         DEC  BL            ; decrementa contador
         JNZ  0105          ; se contador maior que zero continua
         INT  20            ;continua na próxima página

: 0200   PUSH CX
         PUSH DX
         MOV  CX, 0FFF          ; se necessário ajustar o "delay" para o
         MOV  DX, FFFF          ; "clock" do microcomputador do
         DEC  DX                ; laboratório trocando os valores
         JNZ  0208              ; carregados nos registradores
         DEC  CX                ;"CX e DX"
         JNZ  0205
         POP  DX
         POP  CX
         RET
```

2.58 Projeto 1

→ Escrever um programa que possibilite escolher uma, dentre três seqüências de acendimento diferentes para os LED's da placa.

→ Cada seqüência deve ser escolhida através de uma tecla diferente previamente definida.

→ Esta seqüência deve ser executada durante um tempo determinado.

→ Incluir uma quarta tecla para o caso de saída do programa (tecla de escape).

→ "DICAS PARA O PRIMEIRO PROJETO"

1. Primeiramente escrever o algoritmo do programa, envolvendo o programa principal e todas as sub-rotinas.

2. No papel, transcrever para o assembly inicialmente todas as sub-rotinas e só então o programa principal (essa forma de programação contribui para uma maior clareza do problema ajudando também na detecção de erros).

3. Como sugestão de uma boa solução para esse projeto, a estrutura final deve possuir as seguintes características:
   - Uma ou mais sub-rotinas de pausa, para controlar a freqüência de acendimento dos LED's.
   - Contadores para temporizar cada efeito escolhido.
   - Uma sub-rotina para cada efeito.
   - O acesso a essas sub-rotinas será definido por comparação dos bytes ASCII das teclas acionas.
   - Utilizar instruções PUSH e POP quando necessário.

4. Após digitar o programa principal, digitar também todas as sub-rotinas seqüencialmente, criando blocos de instruções não colados, mas sempre separados de múltiplos de 3 bytes (assim será mais fácil introduzir desvios no programa, para eventuais alterações sem a necessidade de uma nova digitação integral), isso é conseguido digitando-se instruções "NOP" entre os blocos (instrução de apenas um byte e que não realiza nenhuma função).

5. Gravar o programa e executa-lo a partir do DOS.

6. Esta estrutura, muito se aproxima, da maioria dos projetos de controle dessa disciplina.

7. Apresentar o resultado para o Professor da disciplina.

2.59 Projeto 2

→ Alterar o programa anterior incluindo a apresentação de mensagens na tela durante a sua execução.

→ Apresentar o resultado para o Professor da disciplina.

2.60 Projeto 3

→ Escrever um programa para temporizar um evento a cada 15 segundos (piscar de um LED na porta paralela ou caractere enviado para a tela)

→ Nesta primeira solução usar uma sub-rotina de delay ajustada experimentalmente e/ou calculada levando-se em conta o número de ciclos de

máquina de cada instrução, a quantidade de loops executados nesta e o tempo de execução do programa principal.

→ Apresentar o resultado para o Professor da disciplina.

2.61 Projeto 4

→ Repetir o projeto anterior porém tendo como solução a leitura do relógio interno do microcomputador.

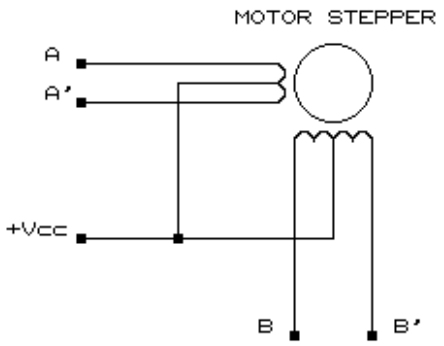
→ Como sugestão é fornecido o seguinte algoritmo simplificado:

1. Ler relógio (segundos).
2. Somar 0Fh ao valor dos segundos.
3. Se soma > 2Ch (2Ch = 44 , 59 – 15 = 44) subtrair 3Ch (= 60) criando o "valor referência".
4. Ler relógio
5. Comparar valor lido com o "valor referência".
6. Se menor vá para 4.
7. Ler relógio.
8. Piscar LED.
9. Testar tecla de saída.
10. Vai para 2.

→ Apresentar o resultado para o Professor da disciplina.

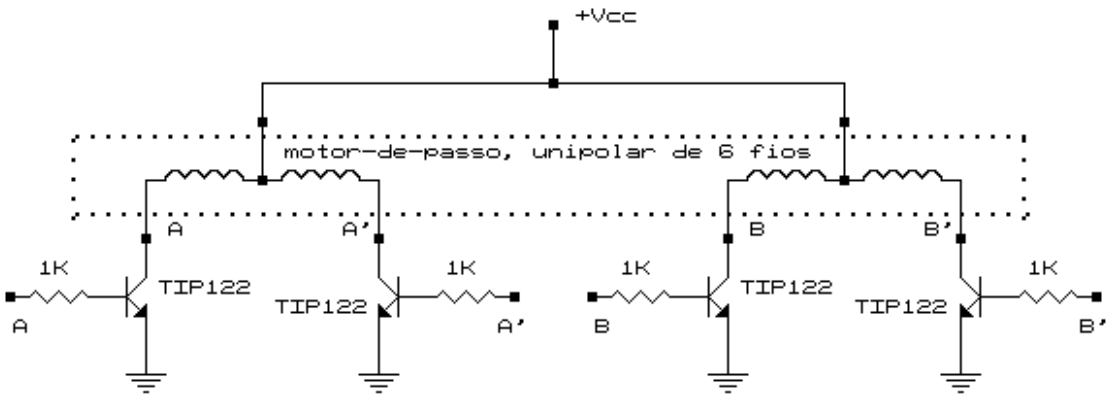
2.62 Exemplo de controle de um motor de passo.

→ A figura mostra o símbolo de um motor de passo genérico.



→ Onde A, A', B e B' são chamadas fases do motor.

→ A seguir é mostrado o esquema eletrônico de um circuito de "drive" genérico, para um motor unipolar de seis fios, sem mostrar os acopladores ópticos (esse esquema deverá ser adaptado ao motor utilizado).



→ Tabela verdade de controle do conjunto "drive" + acopladores ópticos + motor de passo:

→ Observação:
As fases A, B, A' e B' são acionadas pelos bits D3, D2, D1 e D0 respectivamente (conforme ligações do cabo utilizado).

	Fase A	Fase B	Fase A'	Fase B'	Byte
Passo 1	1	1	1	0	#Eh
Passo 2	1	1	0	1	#Dh
Passo 3	1	0	1	1	#Bh
Passo 4	0	1	1	1	#7h

→ Para fazermos o motor girar no sentido horário devemos seguir a tabela anterior na ordem apresentada (..., passo 1, passo 2, passo 3, passo4, passo 1, passo 2, ...) e para invertemos o sentido de rotação, devemos também inverter o sentido de leitura da tabela (..., passo 4, passo 3, passo 2, passo 1, passo 4, passo 3, ...).

→ O motor utilizado possui uma resolução de $1,8^{o}$/passo ou 200 passos/revolução assim, para esse programa exemplo, optaremos por um controle de três voltas, portanto temos 200 x 3 = 600 passos (em decimal) que equivale a 258h.

→ O intervalo de tempo entre cada passo controla a velocidade (maior velocidade equivale a um menor intervalo de tempo) sendo portanto necessária, a execução de uma sub-rotina de pausa entre cada um dos passos. O valor dessa pausa depende das

características de velocidade do microcomputador onde o programa está sendo executado e tem um limite inferior que equivale a máxima velocidade do motor, para este tipo de "drive" utilizado; sendo que após esse valor o motor começa a perder passos e assim tem o seu torque drasticamente diminuído, precisão comprometida e até em um caso extremo, o sentido de rotação invertido.

→ Conectar o motor (ou a placa de LED's) no microcomputador.

→ Digitar o programa exemplo:

```
: 0100  MOV  DX, 0378
        MOV  CX, 0258

        MOV  AL, EE
        OUT  DX, AL
        CALL 0150
        DEC  CX
        JZ   012C
        MOV  AL, DD
        OUT  DX, AL
        CALL 0150
        DEC  CX
        JZ   012C

        MOV  AL, BB
        OUT  DX, AL
        CALL 0150
        DEC  CX
        JZ   012C

        MOV  AL, 77
        OUT  DX, AL
        CALL 0150
        DEC  CX
        JZ   012C

        JMP  0106

        INT  20


: 0150  PUSH CX
        PUSH DX
        MOV  CX, 04FF     ;Ajustar para a velocidade do
        MOV  DX, 04FF     ;microcomputador utilizado.
        DEC  DX
        JNZ  0158         ;continua...
```

```
DEC   CX
JNZ   0155
POP   DX
POP   CX
RET
```

→ Executar e analisar o funcionamento do programa anterior.

→ Salvar o programa.

→ Este não o único formato de um programa para controlar um motor de passo (nem o mais elegante ou eficiente), porém tem como vantagem uma estrutura bastante didática.

2.63 Exercício.

→ Executar o programa anterior alterando a velocidade e o sentido de rotação.

→ Testar a velocidade limite do motor usado.

2.64 Exercício.

→ Modifique o programa anterior para utilizar as instruções de "rotação de bytes" (consultar a tabela de instruções), para obter um programa menor e mais bem elaborado.

→ Salvar o programa.

2.65 Projeto 5.

→ Escrever um programa que possibilite comandar, através do teclado, o sentido de rotação do motor e a sua velocidade além de enviar mensagens correspondentes para a tela.

→ Este programa deve possuir uma tecla de escape e/ou um contador de revoluções.

→ Salvar o programa.

→ Apresentar o resultado para o Professor da disciplina.

2.66 Projeto 6.

→ Escrever um programa que faça a leitura das chaves de fim de curso do kit de motor de passo utilizado no laboratório e execute movimentos com as seguintes características:

1. Inicie o movimento, em um determinado sentido, na máxima velocidade.
2. Ao se aproximar da chave de fim de curso, diminua de velocidade.
3. Ao acionar a chave de fim de curso, inverta o sentido de rotação agora, com máxima velocidade.
4. Repita o item 2, para esse sentido de movimento.
5. Entre em "loop".

→ Este programa deve possuir uma tecla de escape e/ou um contador de revoluções.

→ Salvar o programa.

→ Apresentar o resultado para o Professor da disciplina.

2.67 Projeto 7.

→ Uma outra forma de controlarmos um motor de passo é a sua utilização no modo chamado "meio passo" que possui uma maior precisão de movimento porém um menor torque e uma menor velocidade máxima.

→ Esse modo de operação é conseguido através da seguinte tabela:

	Fase A	Fase B	Fase A'	Fase B'	Byte
Passo 1	1	1	1	0	#Eh
Passo 2	1	1	0	0	#Ch
Passo 3	1	1	0	1	#Dh
Passo 4	1	0	0	1	#9h
Passo 5	1	0	1	1	#Bh
Passo 6	0	0	1	1	#3h
Passo 7	0	1	1	1	#7h
Passo 8	0	1	1	0	#6h

→ Modificar um dos programas dos projetos anteriores para que o motor trabalhe no modo "meio passo".

→ Salvar o programa.

→ Apresentar o resultado para o Professor da disciplina.

CAPÍTULO 3 – PRÁTICA COM O TASM

3.1 O Turbo Assembler – Introdução.

Para que possamos criar um programa, neste novo ambiente, precisamos usar algumas novas ferramentas:

Primeira:
Um editor de texto, para criar o programa fonte.

Segunda:
Um montador, isto é, um programa que irá transformar o fonte em um programa objeto.

Terceira:
Um "linker" (ligador) que irá gerar o programa executável a partir do programa objeto.

O editor pode ser qualquer um disponível (vamos usar o NE – Norton Editor, escolhido pela sua simplicidade e característica de gravação do texto sempre em caracteres ASCII).

O montador será o TASM, Turbo Assembler da Borland.

O "linker" será o TLINK, também da Borland.

Devemos criar os programas fonte com a extensão ".ASM" (obrigatório) para que o TASM reconheça e o transforme no programa objeto, sendo este um "formato intermediário", assim chamado porque ainda não é um programa executável e tão pouco um programa fonte.

O "linker" gera a partir de um programa objeto (com extensão ".OBJ"), ou da combinação de vários deles, um programa executável, cuja extensão é normalmente ".EXE" (ao contrário do DEBUG), embora possa ser ".COM", dependendo da forma como este for montado e ligado.

Os arquivos ".EXE" são organizados em múltiplos segmentos individuais e separados, enquanto os arquivos ".COM" contem somente um segmento, o qual inclui todos os dados e códigos das instruções.

Há vantagens em ambos tipos de arquivos. Por exemplo, os de estrutura ".COM" evitam o tempo de carga necessário para a identificação de segmentos separados, enquanto os arquivos ".EXE" podem acomodar aplicações muito maiores.

3.2 Diretivas – Conceitos básicos.

Para construirmos os programas com o TASM, devemos estruturar o fonte de uma forma diferenciada da empregada no programa DEBUG. Para isso usamos "pseudo-instruções" ou "diretivas" que a princípio são instruções utilizadas pelo compilador e "linker" e não são executadas pelo processador do sistema.

As diretivas "assembly" mais importantes usadas no TASM são:

DOSSEG

Informa ao "linker" (ligador) que os segmentos devem ser agrupados seguindo a mesma ordem adotada pelo sistema operacional.

.MODEL SMALL

Define o meio de memória a usar em nosso programa. Todo o código estará em um segmento, enquanto os dados estarão em outro. Todos os desvios e referências à memória são do tipo "Near" (dentro do segmento) isto é, afetarão apenas o IP.

.DATA

Marca o início do segmento de dados, onde todas as variáveis, tabelas e mensagens do programa devem ser colocadas.

.CODE

Marca o início do segmento de código do programa. No caso de o programa ter mais de um segmento de código, deve-se especificar um nome. Essa diretiva pode aparecer diversas vezes no programa, pois o montador unirá todos os diferente blocos com o mesmo nome para formar um único segmento de código.

.STACK

Reserva espaço de memória para as instruções de programa na pilha. Concatena todos os segmentos com o mesmo nome para formar o segmento de pilha. Na carga do programa, o registrador SS estará automaticamente apontando para esse segmento, e o SP estará posicionado no fim.

PUBLIC

Concatena todos os segmentos com o mesmo nome para formar um único segmento contínuo.

PROC

Declaração usada para definir um bloco de instruções.

ENDP

Declaração que termina o bloco de instruções definido por PROC.

END

Marca o fim do programa-fonte. Tudo o que for colocado após esse ponto será ignorado pelo montador.

EQU

Usada para criar equivalências, atribuindo valores a símbolos

DB

Usada para dados, aloca espaço do tamanho de um byte para cada elemento definido. Pode ser usada para criar variáveis, tabelas e mensagens.

DW

Usada para dados, aloca espaço do tamanho de dois para cada elemento definido. Pode ser usada para criar variáveis, tabelas e mensagens.

3.3 Segmentos

A arquitetura dos processadores das famílias Intel 80x86, 80x88 e Pentium, força-nos a usar segmentos de memória para gerenciar a informação sendo o tamanho destes segmentos de 64Kbytes.

A justificativa do uso destes segmentos está relacionada ao fato do maior valor numérico possível de ser diretamente processado, (limitado a capacidade dos registradores de 16 bits - característica original dos processadores de terceira geração) ser uma palavra de 16 bits (tamanho dos registradores); não sendo assim possível, o acesso a mais de 65536 endereços (64 Kbytes) da memória, usando-se para endereçamento, apenas um destes registradores internos.

Sendo a memória do PC dividida em grupos de segmentos, cada qual com no máximo, 65536 endereços (64 Kbytes), assim podemos usar um registrador exclusivo para endereçar o início de um determinado segmento, e ainda acessar uma posição desejada dentro desse, usando-se outro registrador, possibilitando dessa maneira, o acesso a quantidades muito maiores de memória.

Desta forma, para que o montador seja capaz de gerenciar os dados, se faz necessário que cada informação ou instrução se encontre na área correspondente ao seu segmento. O endereço do segmento é fornecido ao montador pelos registradores “DS, ES, SS e CS” escolhidos conforme o tipo da informação.

Observando como uma instrução é apresentada no DEBUG:

1CB0:0102 MOV AX,BX

O primeiro número 1CB0, corresponde ao segmento de memória que está sendo usado, o segundo é uma referência ao endereço no interior do segmento, é um deslocamento dentro deste, chamado “offset”.

O modo usado para indicar ao montador quais segmentos são usados é empregando as diretivas .CODE, .DATA e .STACK.

O montador ajusta o tamanho dos segmentos (máximo de 64 Kbytes) tomando como base o número de bytes que cada instrução "assembly" precisa, pois seria um desperdício de memória, usar segmentos inteiros para armazenar pequenos programas. Por exemplo, se um programa necessita de apenas 10Kbytes para armazenar dados, o segmento de dados será apenas de 10Kbytes e não de 64Kbytes.

3.4 Construção de um programa "passo-a-passo" em assembly.

Leia atentamente todas as explicações.

→ Primeiro passo:

Use qualquer editor de texto para criar o programa fonte. Entre com as seguintes linhas do exemplo1.asm (nome com no máximo 8 caracteres e sem espaço).

```
;Atenção: use  ;  para fazer comentários em programas "assembly".
;Para o compilador é indiferente o uso de letras maiúsculas ou minúsculas na escrita
;das instruções.
;É muito importante manter uma boa estética durante a digitação do programa
;fonte, pois assim ficará mais fácil o seu futuro estudo e/ou a procura de eventuais
;erros. Normalmente emprega-se um "tab" entre o primeiro e o segundo token
;(símbolo) de cada instrução e digita-se o texto com as instruções alinhadas linha-
;a-linha, a uma certa distância do lado esquerdo da tela (o que facilita a colocação
;dos "labels" para os desvios).

;O TASM possibilita a manipulação de valores numéricos das formas mostradas nos
;exemplos seguintes:

;       MOV   AL, 55        ;o valor 55 está em decimal.
;       MOV   AL, 55h       ;o valor 55h está em hexadecimal.
;       MOV   AL, 5Fh       ;o valor 5F está em hexadecimal (a ausência do "h" no
;                           ;"token", leva a um erro de compilação).
;       MOV   AL, 0F5h      ;o valor F5h está em hexadecimal más só é
;                           ;reconhecido como um valor numérico se iniciar-se
;                           ;com um caractere numérico (a ausência do "0" no
;                           ;token, leva a um erro de compilação).
;       MOV   AL, '@'       ;o token '@' refere-se ao código ASCII do caractere
;                           ;@ isto é 40h (vide tabela).
```

```
;exemplo1.asm

.MODEL SMALL      ;modelo de memória escolhido
.STACK            ;espaço de memória para instruções do programa na pilha
.CODE             ;as linhas seguintes são instruções do programa

MOV  AH, 01h      ;move o valor 01h para o registrador ah
MOV  CX, 07h      ;move o valor 07h para o  registrador cx
INT  10h          ;interrupção 10h
MOV  AH,4Ch       ;move o valor 4ch para o registrador ah
INT  21h          ;interrupção 21h

END               ;finaliza o código do programa
```

→ Segundo passo:

Salvar o arquivo com o seguinte nome: exemplo1.asm (é necessário a extensão ".asm").

Não esquecer de salvá-lo no formato ASCII (caso exista essa opção no editor de texto utilizado).

→ Terceiro passo:

Usar o programa TASM para construir o programa objeto, digitando no sinal de "prompt" do DOS o seguinte comando:

```
C:\>tasm exemplo1.asm
```

Caso não existam erros na compilação, são exibidas na tela, as seguintes mensagens:

```
Turbo Assembler  Version 2.0  Copyright (c) 1988, 1990 Borland International
Assembling file:   exemplo1.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  471k
C:\>
```

Observação:

O TASM só pode criar programas no formato ".OBJ", que ainda não podem ser executados.

→ Quarto passo:

Usar o programa TLINK para criar o programa executável digitando no sinal de "prompt" do DOS o seguinte comando:

C:\>tlink exemplo1.obj

Onde, "exemplo1.obj" é o nome do programa intermediário, ".OBJ". O comando acima gera diretamente o arquivo com o nome do programa intermediário e a extensão ".EXE". É opcional a colocação da extensão ".OBJ" no comando.

Caso não existam erros durante o processo de "linker", é exibida na tela a seguinte mensagem:

Turbo Link Version 3.0 Copyright (c) 1987, 1990 Borland International
C:\>

→ Quinto passo:

Executar o programa criado digitando seu nome (não é necessária a extensão) diretamente no sinal de "prompt" do DOS:

C:\>exemplo1

→ Este programa "assembly" muda o tamanho do cursor no DOS.

→ Para voltar ao cursor normal, retorne ao windows e após novamente ao DOS.

3.5 Exemplos de programas.

Nesta seção são mostrados vários exemplos de programas para serem montados e ligados com o TASM e o TLINK da Borland, com o objetivo de familiarizar o leitor/aluno com as características mais usuais desse ambiente de programação.

Leia atentamente as explicações fornecidas, analise o funcionamento de todos os programas, principalmente os detalhes relacionados as diferenças estruturais e diretivas empregadas.

Digite, salve, compile ligue e execute alguns dos programas fornecidos, lembrando que muitos são parecidos com os executados via DEBUG.

Por uma questão de tempo fica opcional, a digitação das explicações que acompanham os cabeçalhos e instruções dos exemplos sendo estas porém, importantes para a documentação dos programas criados nos projetos propostos.

```
;exemplo2.asm

.MODEL SMALL
.STACK
.CODE

MOV  AH, 2h          ;move o valor 2h para o registrador ah
MOV  DL, 2Ah         ;move o valor 2ah para o registrador dl
                     ;(é o valor ASCII do caractere *)
INT  21h             ;interrupção 21h
MOV  AH, 4ch         ;função 4ch, sai para o sistema operacional
INT  21h             ;interrupção 21h

END                  ;finaliza o programa
```

→ Este programa imprime o caractere * na tela.

```
;exemplo3.asm

;Atenção para o uso de LABEL's seguidos de : (dois pontos) para indicar os
;endereços de desvios.

.MODEL SMALL
.STACK
.CODE

        MOV  AH, 1h      ;Função 1 do DOS
        INT  21h         ;lê o caractere e retorna o código ASCII ao registrador AL
        MOV  DL, AL      ;move o código ASCII para o registrador DL
        SUB  DL, 30h     ;subtrai de 30h para converter a um dígito de 0 a 9
        CMP  DL, 9h      ;compara se o dígito está entre 0 e 9
        JLE  digit1      ;se verdadeiro obtém o primeiro número (4 bits)
        SUB  DL, 7h      ;se falso, subtrai de 7h para converter a uma letra (A a F)
digit1: MOV  CL, 4h      ;prepara para multiplicar por 16
        SHL  DL, CL      ;multiplica para converter dentro dos 4 bits mais altos
        INT  21h         ;obtém o próximo caractere
        SUB  AL, 30h     ;repete a operação de conversão
        CMP  AL, 9h      ;compara o valor 9h com o conteúdo do registrador AL
        JLE  digit2      ;se verdadeiro, obtém o segundo dígito
        SUB  AL, 7h      ;se falso, subtrai de 7h
digit2: ADD  DL, AL      ;adiciona o segundo dígito
        MOV  AH, 4Ch     ;função 4Ch do DOS ("exit")
        INT  21h         ;interrupção 21h

        END              ;finaliza o programa
```

→ Este programa lê dois caracteres e os imprime na tela

```
        ;exemplo4.asm

        ;Atenção para o uso de caracteres em decimal (sem "h" acompanhando o número), e
        ;caracteres ASCII (entre 'aspas')  representando o código correspondente.

        ;Atenção para o uso obrigatório das diretivas PROC e ENDP para iniciar e finalizar
        ;um procedimento ou sub-rotina.

        ;Atenção para o uso do caractere _ ("underline") para o nome de procedimentos;
        ;não é permitido o uso de espaço para tal.

        .MODEL SMALL
        .STACK
        .CODE

PRINT_A_J       PROC

        MOV DL, 'A'             ;move o código ASCII (byte) do caractere A para o
                                ;registrador DL
        MOV CX, 10              ;move o valor decimal 10 para o registrador CX
                                ;este valor é usado para fazer laço com 10 interações
PRINT_LOOP:
        CALL WRITE_CHAR         ;Imprime o caractere em DL
        INC DL                  ;Incrementa o valor do registrador DL
        LOOP PRINT_LOOP         ;Laço para imprimir 10 caracteres
        MOV   AH, 4Ch           ;Função 4Ch, para sair ao DOS
        INT     21h             ;Interrupção 21h

        PRINT_A_J       ENDP    ;Finaliza o procedimento

        WRITE_CHAR      PROC

        MOV   AH, 2h            ;Função 2h, imprime caractere
        INT     21h             ;Imprime o caractere que está em DL
        RET                     ;Retorna o controle ao procedimento que chamou

        WRITE_CHAR      ENDP    ;Finaliza o procedimento

        END  PRINT_A_J          ;Finaliza o programa
```

→ Este programa mostra os caracteres ABCDEFGHIJ na tela.

```
;.....
;       exemplo5.asm
;       Atenção para os cabeçalhos
;       Atenção para o uso  da diretiva PUBLIC
;.....

        .MODEL SMALL
        .STACK
        .CODE

        TEST_WRITE_HEX    PROC

        MOV   DL, 3Fh                   ;Move o valor 3Fh para o registrador DL
        CALL  WRITE_HEX                 ;Chama a sub-rotina
        MOV   AH, 4CH                   ;Função 4Ch
        INT     21h                     ;Retorna o controle ao DOS

        TEST_WRITE_HEX ENDP             ;Finaliza o procedimento

        PUBLIC WRITE_HEX
;.....
;       Este procedimento converte para hexadecimal o byte
;       armazenado no registrador DL e mostra o dígito
;       Use:WRITE_HEX_DIGIT
;.....

        WRITE_HEX    PROC

        PUSH  CX                        ;envia para  pilha o valor do registrador CX
        PUSH  DX                        ;envia para pilha o valor do registrador DX
        MOV   DH, DL                    ;move o valor do registrador DL para o
                                        ;registrador DH
        MOV   CX, 4                     ;move o valor 4 para o registrador CX
        SHR    DL, CL
        CALL  WRITE_HEX_DIGIT           ;mostra na tela o primeiro número hexadecimal
        MOV   DL, DH                    ;move o valor do registrador DH para o
                                        ;registrador DL
        AND   DL, 0Fh
        CALL  WRITE_HEX_DIGIT           ;mostra na tela o segundo número hexadecimal
        POP    DX                       ;retorna da pilha o valor do registrador DX
        POP    CX                       ;retorna da pilha o valor do registrador CX
        RET                             ;retorna o controle ao procedimento que
                                        ;chamou

        WRITE_HEX  ENDP

        PUBLIC WRITE_HEX_DIGIT
```

```
;.....
;       Este procedimento converte os 4 bits mais baixos do registrador DL
;       para um número hexadecimal e o mostra na tela do computador
;       Use: WRITE_CHAR
;.....

        WRITE_HEX_DIGIT    PROC

        PUSH  DX                        ;envia para a pilha o valor de DX
        CMP   DL, 10                    ;compara se o número de bits é menor
                                        ;do que 10
        JAE     HEX_LETTER              ;se não, salta para HEX_LETER
        ADD   DL, "0"                   ;se sim, converte para número
        JMP Short       WRITE_DIGIT     ;escreve o caractere
HEX_LETTER:
        ADD   DL, "A"-10                ;converte um caractere para hexadecimal
WRITE_DIGIT:
        CALL  WRITE_CHAR                ;imprime o caractere na tela
        POP     DX                      ;Retorna o valor inicial do registrador DX
                                        ;para o registrador DL
        RET                             ;Retorna o controle ao procedimento que
                                        ;chamou

        WRITE_HEX_DIGIT   ENDP

    PUBLIC WRITE_CHAR
;.....
;       Este procedimento imprime um caractere na tela usando o DOS
;.....

        WRITE_CHAR   PROC

        PUSH  AX                        ;Envia para pilha o valor do registrador AX
        MOV   AH, 2h                    ;Função 2h
        INT     21h                     ;Interrupção 21h
        POP     AX                      ;Extrai da pilha o valor de AX
        RET                             ;Retorna o controle ao procedimento que
                                        ;chamou

        WRITE_CHAR   ENDP

        END TEST_WRITE_HEX              ;Finaliza o programa
```

→ Este programa faz a conversão de bytes em caracteres correspondentes impressos na tela do monitor.

```
;.....
;       exemplo6.asm
;.....

        .MODEL SMALL
        .STACK
        .CODE

        TEST_WRITE_DECIMAL    PROC

        MOV   DX, 12345             ;Move o valor decimal 12345 para o
                                    ;registrador DX
        CALL WRITE_DECIMAL          ;Chama o procedimento
        MOV   AH, 4CH               ;Função 4Ch
        INT     21h                 ;Interrupção 21h

        TEST_WRITE_DECIMAL ENDP  ;Finaliza o procedimento

        PUBLIC WRITE_DECIMAL
;.....
;       Este procedimento escreve um número de 16 bits como um número
;       sem sinal em notação decimal
;       Use: WRITE_HEX_DIGIT
;.....

        WRITE_DECIMAL    PROC

        PUSH  AX                    ;Envia para  pilha o valor do registrador AX
        PUSH  CX                    ;Envia para  pilha o valor do registrador CX
        PUSH  DX                    ;Envia para  pilha o valor do registrador DX
        PUSH  SI                    ;Envia para  pilha o valor do registrador SI
        MOV   AX, DX                ;move o valor do registrador DX para AX
        MOV   SI, 10                ;move o valor 10 para o registrador SI
        XOR   CX, CX                ;zera o registrador CX
NON_ZERO:
        XOR   DX, DX                ;zera o registrador DX
        DIV     SI                  ;divisão entre SI
        PUSH  DX                    ;Envia para a pilha o valor do registrador DX
        INC     CX                  ;incrementa CX
        OR      AX, AX              ;não zero
        JNE     NON_ZERO            ;salta para NON_ZERO
WRITE_DIGIT_LOOP:
        POP     DX                  ;Retorna o valor em modo reverso
        CALL  WRITE_HEX_DIGIT       ;Chama o procedimento
        LOOP WRITE_DIGIT_LOOP       ;loop
END_DECIMAL:
        POP     SI                  ;retorna da pilha o valor do registrador SI
```

```
        POP    DX                  ;retorna da pilha o valor do registrador DX
        POP    CX                  ;retorna da pilha o valor do registrador CX
        POP    AX                  ;retorna da pilha o valor do registrador AX
        RET                        ;retorna o controle ao procedimento que
                                   ;chamou

        WRITE_DECIMAL   ENDP       ;Finaliza o procedimento

        PUBLIC WRITE_HEX_DIGIT
;.....,
;       Este procedimento converte os 4 bits mais baixos do registrador DL
;       num número hexadecimal e os imprime
;       Use: WRITE_CHAR
;.....

        WRITE_HEX_DIGIT    PROC

        PUSH  DX                   ;Envia para  pilha o valor do registrador DX
        CMP   DL, 10               ;Compara o valor 10 com o valor do
                                   ;registrador DL
        JAE     HEX_LETTER         ;se não, salta para HEX_LETER
        ADD   DL, "0"              ;se sim, converte em dígito numérico
        JMP Short       WRITE_DIGIT    ;escreve o caractere
HEX_LETTER:
        ADD   DL, "A"-10           ;converte um caractere para um número
                                   ;hexadecimal
WRITE_DIGIT:
        CALL  WRITE_CHAR           ;mostra o caractere na tela
        POP    DX                  ;Retorna o valor inicial para o registrador DL
        RET                        ;Retorna o controle ao procedimento que
                                   ;chamou
        WRITE_HEX_DIGIT   ENDP

        PUBLIC WRITE_CHAR
;.....
;       Este procedimento imprime um caractere na tela usando uma função DOS
;.....

        WRITE_CHAR   PROC

        PUSH  AX                   ;Envia para  pilha o valor do registrador AX
        MOV   AH, 2h               ;Função 2h
        INT     21h                ;Interrupção 21h
        POP    AX                  ;Retorna da pilha o valor inicial do registrador
                                   ;AX
        RET                        ;Retorna o controle ao procedimento que
                                   ;chamou
```

```
        WRITE_CHAR   ENDP

        END TEST_WRITE_DECIMAL      ;finaliza o programa
```

→ Este programa mostra na tela os números 12345

```
;.....
;         exemplo7.asm
;.....

        .MODEL SMALL
        .STACK
        .CODE

        PRINT_ASCII     PROC

        MOV  DL, 00h                ;move o valor 00h para o registrador DL
        MOV  CX, 255                ;move o valor decimal 255 para o registrador
                                    ;CX usado para fazer um laço com 255
                                    ;interações
PRINT_LOOP:
        CALL WRITE_CHAR             ;Chama o procedimento que imprime
        INC  DL                     ;Incrementa o valor do registrador DL
        LOOP PRINT_LOOP             ;"Loop" para imprimir 10 caracteres
        MOV  AH, 4Ch                ;Função 4Ch
        INT  21h                    ;Interrupção 21h

        PRINT_ASCII     ENDP        ;Finaliza o procedimento

        WRITE_CHAR      PROC

        MOV  AH, 2h                 ;Função 2h para imprimir um caractere
        INT  21h                    ;Imprime o caractere que está em DL
        RET                         ;Retorna o controle ao procedimento que
                                    ;chamou

        WRITE_CHAR      ENDP        ;Finaliza o procedimento

        END  PRINT_ASCII            ;Finaliza o programa
```

→ Este programa mostra na tela o valor dos 256 caracteres do código ASCII.

```
;*************************************************************************
;       exemplo8.asm
;       Descrição:      Exemplo  do uso do TASM com mensagens na tela e aplicação
;                       das diretivas:   .DATA, EQU, DB e DW
;*************************************************************************
;nome do programa: exemplo8.asm

        CR      EQU   13                ;ASCII Carriage Return
        LF      EQU   10                ;ASCII Line Feed
        DOSSEG
        .MODEL SMALL
        .STACK
        .DATA

;*************************************************************************
;       REGIÃO DE DEFINIÇÃO DE VARIÁVEIS
;*************************************************************************
CONTADOR DB     00H                     ;estabelece o valor inicial do contador de
                                        ;número de vezes que a tecla é pressionada

;*************************************************************************
;       REGIÃO DE DEFINIÇÃO DE CONSTANTES
;*************************************************************************
MENS_INICIAL DB  cr,lf,lf,'>> PROGRAMA EXEMPLO 8 << '
             DB  cr,lf,lf,'Tecla a (minuscula) para prosseguir'
             DB  cr,lf,lf,'tecla s (minuscula) para sair$'

MENS_FINAL   DB  cr,lf,lf,'>> PROGRAMA FINALIZADO <<'
             DB  cr,lf,lf,'------- TCHAU!! -------$'

MENS_1           DB  cr,lf,lf,'Voce pessionou "a" pela primeira vez$'

MENS_2           DB  cr,lf,lf,'Voce pressionou "a" pela segunda vez$'

MENS_3           DB  cr,lf,lf,'Voce pressionou "a" pela terceira e ultima vez$'

        .CODE
EXEMPLO_8 PROC
        mov     ax, @data               ;seqüência necessária para mensagens na tela
        mov     ds, ax                  ;inicializa o registradores de segmento
        mov     es, ax                  ;DS e CS
        cld                             ;reseta flag de direção

        call    ESCREVE_INICIO

L2:     mov     ah, 08h                 ;leitura de teclado sem eco
        int     21h
```

```
        cmp     al, 61h                 ;verifica tecla "A" minúscula
        jnz     L1
        call    MENSAGENS
        jmp     L2
L1:     cmp     al, 73h                 ;verifica tecla "S" minúscula
        jz      L3
        jmp     L2
L3:     call    ESCREVE_FINAL
        mov     ah, 4ch
        int     21h
EXEMPLO_8 ENDP

;*************************************************************************
;       subrotina:      ESCREVE_INICIO
;                       Escreve mensagem inicial
;*************************************************************************
PUBLIC ESCREVE_INICIO
ESCREVE_INICIO  PROC
        push    ax
        push    dx
        mov     ah, 09h
        lea     dx, MENS_INICIAL        ;carregar o endereço externo
        int     21h                     ;(EA=endereço inicial da mensagem)
        pop     dx                      ;para o registrador
        pop     ax
        ret
ESCREVE_INICIO  ENDP

;*********************************************************
;       subrotina:      ESCREVE_FINAL
;                       Escreve mensagem final
;*********************************************************
PUBLIC ESCREVE_FINAL
ESCREVE_FINAL   PROC
        push    ax
        push    dx
        mov     ah, 09h
        lea     dx, MENS_FINAL          ;carregar o endereço externo
        int     21h                     ;(EA=endereço inicial da mensgem)
        pop     dx                      ;para o registrador
        pop     ax
        ret
ESCREVE_FINAL   ENDP
```

```
;*************************************************************
;       subrotina:      MENSAGENS
;                       Escreve mensagem correspondente ao número de vezes
;                       que a tecla "a"  é pressionada.
;*************************************************************
PUBLIC MENSAGENS
MENSAGENS       PROC
        push    ax
        push    dx
        cmp     CONTADOR, 00h
        jnz     L10
        mov     ah, 09h
        lea     dx, MENS_1              ;carregar o endereço externo
        int     21h                     ;(EA=endereço inicial da mensagem)
                                        ;para o registrador
        inc     CONTADOR
        pop     dx
        pop     ax
        ret

L10:    cmp     CONTADOR, 01h
        jnz     L11
        mov     ah, 09h
        lea     dx, MENS_2              ;carregar o endereço externo
        int     21h                     ;(EA=endereço inicial da mensagem)
                                        ;para o registrador
        inc     CONTADOR
        pop     dx
        pop     ax
        ret

L11:    cmp     CONTADOR, 02h
        jnz     L12
        mov     ah, 09h
        lea     dx, MENS_3              ;carregar o endereço externo
        int     21h                     ;(EA=endereço inicial da mensagem)
                                        ;para o registrador
        inc     CONTADOR
        call    ESCREVE_FINAL
        mov     ah, 4ch
        int     21h

L12:    pop     dx
        pop     ax
        ret
MENSAGENS       ENDP
                END
```

3.6 Projetos com TASM

Escrever novamente alguns dos programas referentes aos projetos (realizados anteriormente no DEBUG) listados a seguir, agora utilizando o TASM como ambiente de programação.

Para conseguir enviar mensagens na tela nos projetos propostos, estudar o exemplo 8 anterior e outro fornecido no capítulo 5 (programa ANA14.ASM) um projeto completo que trabalha com diversas telas, menus etc, concentrando sua atenção no emprego das diretivas “EQU”, “DB” e “DW”.

Projeto 1 (item 2.56)
Projeto 2 (item 2.57)
Projeto 3 (item 2.58)
Projeto 4 (item 2.59)
Projeto 5 (item 2.63)
Projeto 6 (item 2.64)
Projeto 7 (item 2.65)

CAPITULO 4 – PROJETO DE UMA INTERFACE

4.1 Introdução:

A participação em um projeto (mesmo de pequeno porte e com apenas uma finalidade didática) é muito importante para o aprendizado do aluno, pois somente através da montagem e da utilização prática de uma interface montada, ganha-se uma real experiência sobre as dificuldades envolvidas no projeto e na aplicação de um circuito para computador.

Com o objetivo de iniciar os alunos nesta área e com as dificuldades didáticas inerentes da baixa carga horária dos cursos atuais que, inevitavelmente leva-nos a formas mais objetivas de ensinar, é apresentado neste capítulo, o projeto completo de uma placa de interface "multi-uso ou multi I/O", que poderá ser totalmente utilizado ou adaptado com facilidade, às necessidades e interesse do(s) aluno(s).

Este conteúdo tem o objetivo de apenas iniciar o aluno e não portanto a pretensão de tentar esgotar tão vasto assunto.

Inicialmente são apresentadas características de "hardware" do PC relevantes ao projeto, seguido de informações sobre os CI's mais importantes envolvidos e finalmente o esquema eletrônico**.**

Para aplicações que necessitem de mais informações técnicas sobre o hardware do PC, torna-se indispensável consultar publicações especializadas no assunto.

4.2 Mapa de endereçamento de I/O

O conhecimento do mapa de I/O é parte fundamental que qualquer projeto de interface para o microcomputador padrão IBM-PC. A figura seguinte apresenta as diversas faixas de endereços de I/O e seus respectivos dispositivos.

OBS:

A rapidez com que se oferecem placas de expansão para o PC implica na necessidade de uma constante atualização dessa tabela.

Endereços	Dispositivo
de 0000h a 000Fh	Primeiro controlador de DMA (8237)
de 0020h a 0021h	Primeiro controlador de interrupções (8259)
de 0040h a 0043h	Temporizador programável (8253)
de 0060h a 0063h	Interface paralela 8255 (XT)
de 0060h a 0064h	Controlador 8742 (AT)
de 0070h a 0071h	RAM CMOS e registrador de máscara NMI (AT)
de 0080h a 008Fh	Registradores de paginação para DMA
de 00A0h a 00A1h	Segundo controlador de interrupções (AT)
de 00C0h a 00DFh	Segundo controlador de DMA (AT)
de 00F0h a 00FFh	Co-processador aritmético (AT)
de 01F0h a 01FFh	Disco rígido (AT)
de 0200h a 020Fh	Porta de jogos
de 0238h a 023Bh	Mouse de barramento
de 0278h a 027Fh	LPT2
de 02B0h a 02BFh	Vídeo (EGA-VGA)
de 02C0h a 02CFh	Vídeo (EGA-VGA)
de 02D0h a 02DFh	Vídeo (EGA-VGA)
de 02E8h a 02EFh	COM4
de 02F8h a 02FFh	COM2
de 0300h a 031Fh	Reservado para protótipos
de 0320h a 032Fh	Disco rígido (XT)
de 0378h a 037Fh	LPT1
de 03C0h a 03CFh	Vídeo (EGA-VGA)
de 03D0h a 03DFh	Vídeo (CGA)
de 03E8h a 03EFh	COM3
de 03F0h a 03F7h	Disco flexível
de 03F8h a 03FFh	COM1
Acima de 0400h	Projetos PCI

4.3 O SLOT ISA (Industry Standard Architecture) – Apresentação:

Os “Slots” padrão ISA, são conectores de expansão do “bus”de dados, “bus”de endereços, “bus”de controle e alimentação, que permitem a ligação de vários tipos de periféricos à placa da CPU, através de um conector de 62 pinos divididos em dois lados com 31 pinos cada e de outro com 36 pinos, divididos em dois lados com 18 pinos cada.

As placas de periféricos que são ligadas nos “slots” são acessadas através do uso dos endereços reservados para esse fim no mapa de I/O. Esses sinais possuem níveis de tensão compatíveis com TTL (com exceção para as tensões de alimentação) e quando saída capacidade de ativar até duas cargas TTL LOW SCHOTTKY (74LS ...).

PINAGEM DO SLOT ISA – IBM PC

OBS: - Os sinais marcados com ( )' são ativos em "0".
- O simbolo "→" quando direcionado para o retângulo, que representa o SLOT, indica saída, no sentido inverso indica entrada, e o sinal "←→" indica linha bidirecional.
- Os pinos com números iniciados por "A..." e "C..." referem-se ao conhecido "lado dos componentes" da placa e os com números iniciados por "B..." e "D..." ao "lado da solda", sendo sua contagem iniciada sempre em relação ao painel traseiro do computador.

O SLOT original do PC XT

Sinal	pino	pino	Sinal
GND	B1	A1	→ (I/O CH CK)'
RESET DRV →	B2	A2	←→ D7
+5V	B3	A3	←→ D6
IRQ2 ←	B4	A4	←→ D5
-5V	B5	A5	←→ D4
DRQ2 ←	B6	A6	←→ D3
-12V	B7	A7	←→ D2
reservado	B8	A8	←→ D1
+12V	B9	A9	←→ D0
GND	B10	A10	← I/O CH RDY
(MEMW)' →	B11	A11	← AEN
(MEMR)' →	B12	A12	← A19
(IOW)' →	B13	A13	← A18
(IOR)' →	B14	A14	← A17
(DACK3)' →	B15	A15	← A16
DRQ3 ←	B16	A16	← A15
(DACK1)' →	B17	A17	← A14
DRQ1 ←	B18	A18	← A13
(DACK0)' →	B19	A19	← A12
CLOCK →	B20	A20	← A11
IRQ7 ←	B21	A21	← A10
IRQ6 ←	B22	A22	← A9
IRQ5 ←	B23	A23	← A8
IRQ4 ←	B24	A24	← A7
IRQ3 ←	B25	A24	← A6
(DACK2)' →	B26	A26	← A5
TC →	B27	A27	← A4
ALE →	B28	A28	← A3
+5V	B29	A29	← A2
OSC →	B30	A30	← A1
GND	B31	A31	← A0

O SLOT ISA para 286, 386, 486 e Pentium

OBS: Notar as diferenças nos pinos B4, B8, B11 e B12.

Sinal	pino	pino	Sinal
GND	B1	A1	→ (I/O CH CK)’
RESET DRV →	B2	A2	←→ D7
+5V	B3	A3	←→ D6
IRQ9 ←	B4	A4	←→ D5
-5V	B5	A5	←→ D4
DRQ2 ←	B6	A6	←→ D3
-12V	B7	A7	←→ D2
(OWS)’ →	B8	A8	←→ D1
+12V	B9	A9	←→ D0
GND	B10	A10	← I/O CH RDY
(SMEMW)’ →	B11	A11	← AEN
(SMEMR)’ →	B12	A12	← A19
(IOW)’ →	B13	A13	← A18
(IOR)’ →	B14	A14	← A17
(DACK3)’ →	B15	A15	← A16
DRQ3 ←	B16	A16	← A15
(DACK1)’ →	B17	A17	← A14
DRQ1 ←	B18	A18	← A13
(DACK0)’ →	B19	A19	← A12
CLOCK →	B20	A20	← A11
IRQ7 ←	B21	A21	← A10
IRQ6 ←	B22	A22	← A9
IRQ5 ←	B23	A23	← A8
IRQ4 ←	B24	A24	← A7
IRQ3 ←	B25	A24	← A6
(DACK2)’ →	B26	A26	← A5
TC →	B27	A27	← A4
ALE →	B28	A28	← A3
+5V	B29	A29	← A2
OSC →	B30	A30	← A1
GND	B31	A31	← A0

Sinal	pino	pino	Sinal
(MEM CS 16)' ←	D1	C1	→ (SBHE)'
(I/O CS 16)' ←	D2	C2	→ LA23
IRQ10 ←	D3	C3	→ LA22
IRQ11 ←	D4	C4	→ LA21
IRQ12 ←	D5	C5	→ LA20
IRQ15 ←	D6	C6	→ LA19
IRQ14 ←	D7	C7	→ LA18
(DACK0)' →	D8	C8	→ LA17
DRQ0 ←	D9	C9	←→ (MEMR)'
(DACK5)' →	D10	C10	←→ (MEMW)'
DRQ5 ←	D11	C11	←→ D8
(DACK6)' →	D12	C12	←→ D9
DRQ6 ←	D13	C13	←→ D10
(DACK7)' →	D14	C14	←→ D11
DRQ7 ←	D15	C15	←→ D12
+5V	D16	C16	←→ D13
(MASTER)' ←	D17	C17	←→ D14
GND	D18	C18	←→ D15

Descrição dos sinais usados no projeto exemplo:

OBS: Descrições mais detalhadas destes sinais e informações adicionais sobre os outros são encontradas nas literaturas especializadas.

RESET DRV ("Reset Drive"), saída:

Este sinal é mantido ativo durante a seqüência de energização do sistema, por aproximadamente 2 segundos. Além disso, é automaticamente ativado se alguma das fontes sair fora das especificações. É usado para inicializar todos os dispositivos conectados ao barramento de expansão, sendo sincronizado pelo 8284 (ou compatível).

AO-A19, (endereços) saídas:

Estes são os bits usados para endereçar a memória do sistema e os dispositivos de I/O. Eles são gerados pela CPU durante acessos à memória e aos dispositivos de I/O. Durante os ciclos de DMA, eles são gerados pelo controlador de DMA. Com 20 linhas de endereço, pode-se acessar até 1 Mbyte de memória ou, através de instruções de "IN" e "OUT", até 1 K dispositivos de I/O, pois essas instruções emitem endereços pelas linhas A0 até A15. As linhas A16 até A19 não são ativadas durante as operações de I/O.

DO-D7, (dados) bidirecional:

Estas linhas são usadas para transmitir dados entre a CPU e a memória ou dispositivos de I/O. Elas estarão válidas um pouco antes do flanco ascendente (T) dos sinais ((S)MEMW)' e (IOW)'. Normalmente, usam-se o (IOW)' como "strobe" para escrita nos dispositivos de I/O e a linha ((S)MEMW)' como "strobe" de escrita na memória. Durante

as operações de leitura ((IOR)' ou ((S)MEMR))'), os dados devem estar válidos um pouco antes do flanco ascendente (T) de (IOR)' ou ((S)MEMR)'. Nas operações de DMA, essas linhas permitem que o dado trafegue diretamente entre a memória e o dispositivo de I/O.

(IOR)' (" I/O Read"), saída:

Este é um sinal gerado pelo Controlador de Barramento e indica aos dispositivos de I/O que o atual ciclo de barramento é um ciclo de leitura de I/O. O dispositivo endereçado deve responder colocando o dado no barramento pouco antes da subida do sinal (IOR)'. Durante os ciclos de DMA, o sinal (IOR)' é gerado pelo Controlador de DMA. Nestes casos, o endereço presente no barramento diz respeito a uma posição de memória e não a um dispositivo de I/O e o dispositivo de I/O é selecionado não por endereço, mas sim pelo sinal (DACK)'.

(IOW)' (" I/O Write"), saída:

Este é um sinal semelhante ao anterior, exceto por servir para escritas em um dispositivo de I/O. O dispositivo endereçado deve capturar (fazer um "latch") o barramento de dados com o flanco ascendente (T) desse sinal. A mesma afirmação é válida para os ciclos de DMA.

AEN (" Address Enable"), saída:

É gerado pela lógica de controle de DMA e serve para indicar que se está executando um ciclo de DMA. De forma mais precisa, desabilita o barramento que estava de posse da CPU para colocá-lo à disposição do controlador de DMA. Para os dispositivos de I/O conectados ao barramento de expansão, este sinal deve ser usado para desabilitar os decodificadores. Isto impede que um I/O responda erroneamente ao endereço de DMA (pois diz respeito à memória), já que vai estar presente um (IOR)' ou (IOW)'.

+5 VDC:

Regulagem de ±5%, está presente em dois pinos. Alimenta os circuitos lógicos.

+12 VDC:

Regulado para ±5%. Alimenta motor dos acionadores de disco e também a porta serial.

-5 VDC:

Regulado para ±10%. Alimentava memórias antigas que também precisavam de -5V para operarem.

-12 VDC:

Regulado para ±10%. Alimenta porta serial.

GND:

Referência de terra, presente em 3 pinos.

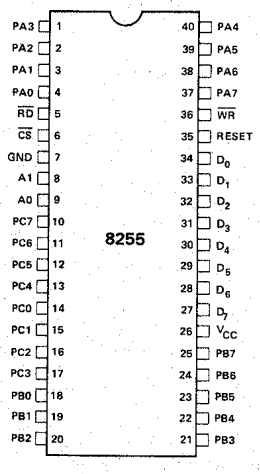
4.4 Principais C.I.'s

A Interface de Comunicação Paralela Programável 8255 (PPI):

A interface paralela 8255A possui três portas, de 8 bits cada, (chamadas A, B e C), que podem ser programadas independentemente como entradas ou saídas. A porta C pode ser programada para se tornar duas portas de 4 bits cada (independentes), formando assim, os chamados GRUPO A (porta A + 4 bits superiores da porta C) e o GRUPO B (porta B + 4 bits inferiores da porta C).

Nas figuras seguintes são apresentadas a sua pinagem, a arquitetura interna e os respectivos bytes de programação.

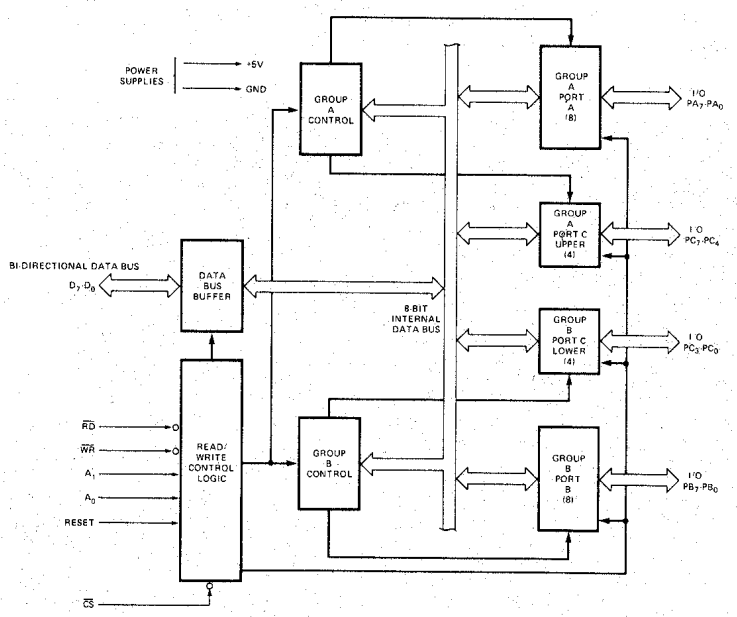
PIN CONFIGURATION

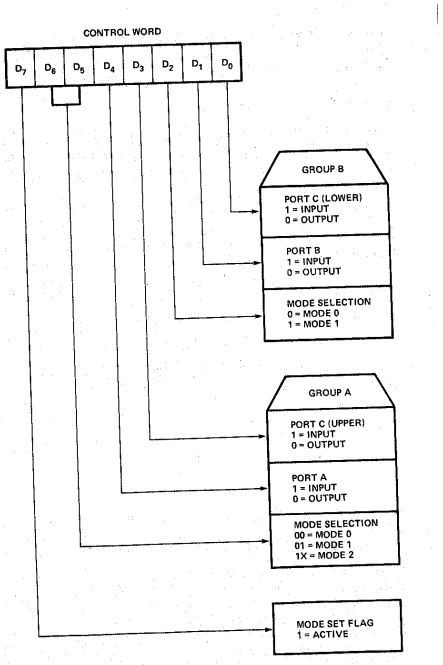


PIN NAMES

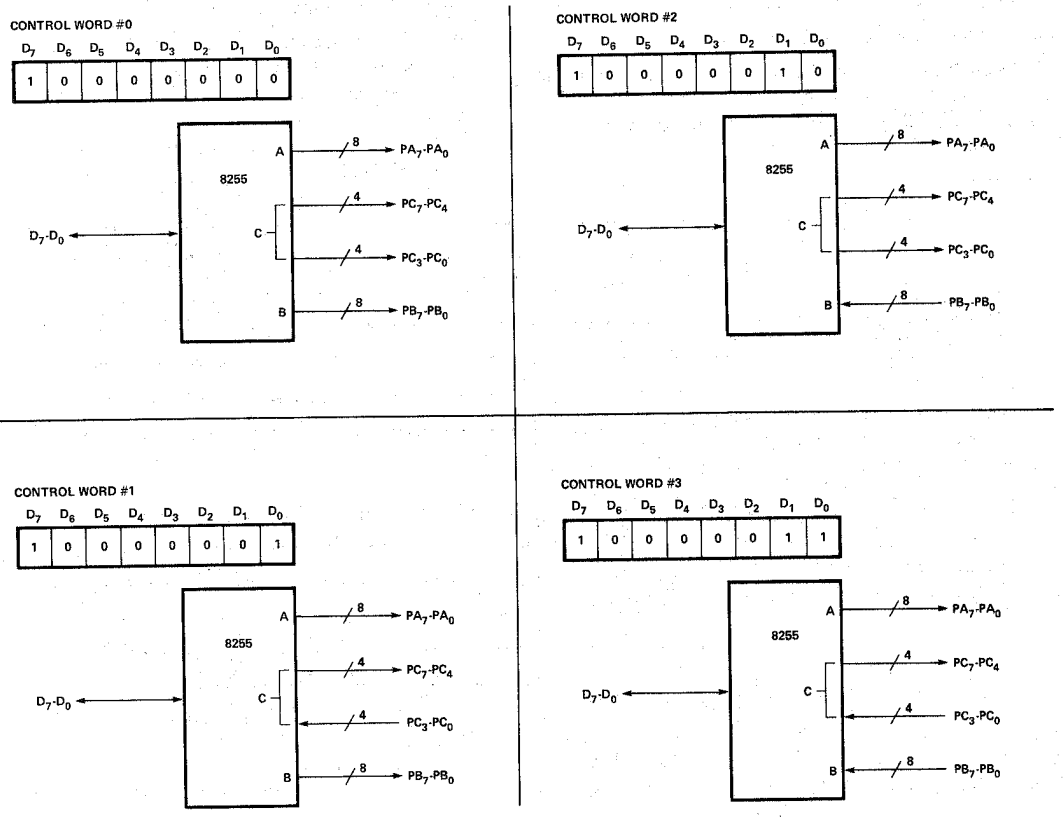
D_7–D_0	DATA BUS (BI-DIRECTIONAL)
RESET	RESET INPUT
$\overline{CS}$	CHIP SELECT
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A0, A1	PORT ADDRESS
PA7-PA0	PORT A (BIT)
PB7-PB0	PORT B (BIT)
PC7-PC0	PORT C (BIT)
V_{CC}	+5 VOLTS
GND	Ø VOLTS

8255 BLOCK DIAGRAM



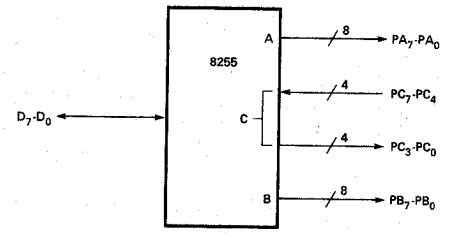


MODE 0 CONFIGURATIONS



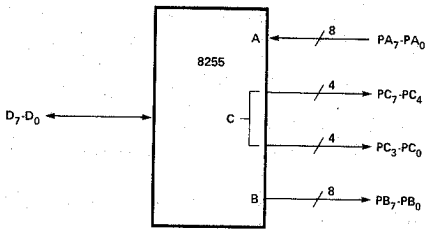
CONTROL WORD #4

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	0



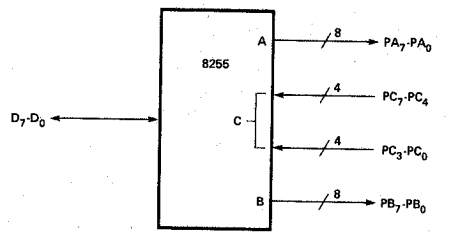
CONTROL WORD #8

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	0



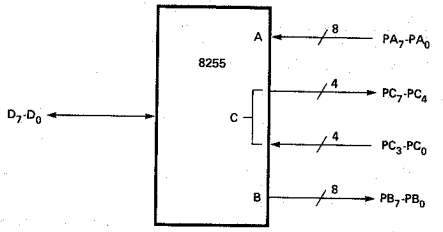
CONTROL WORD #5

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	1



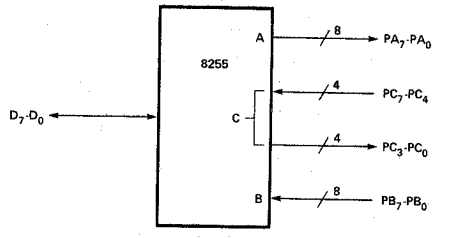
CONTROL WORD #9

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	1



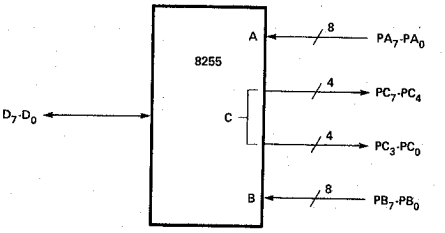
CONTROL WORD #6

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	0



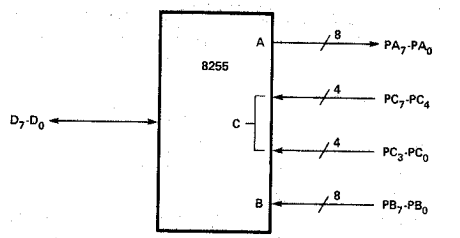
CONTROL WORD #10

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	0



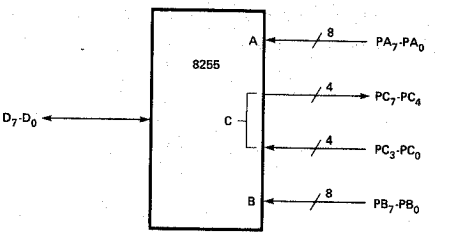
CONTROL WORD #7

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	1

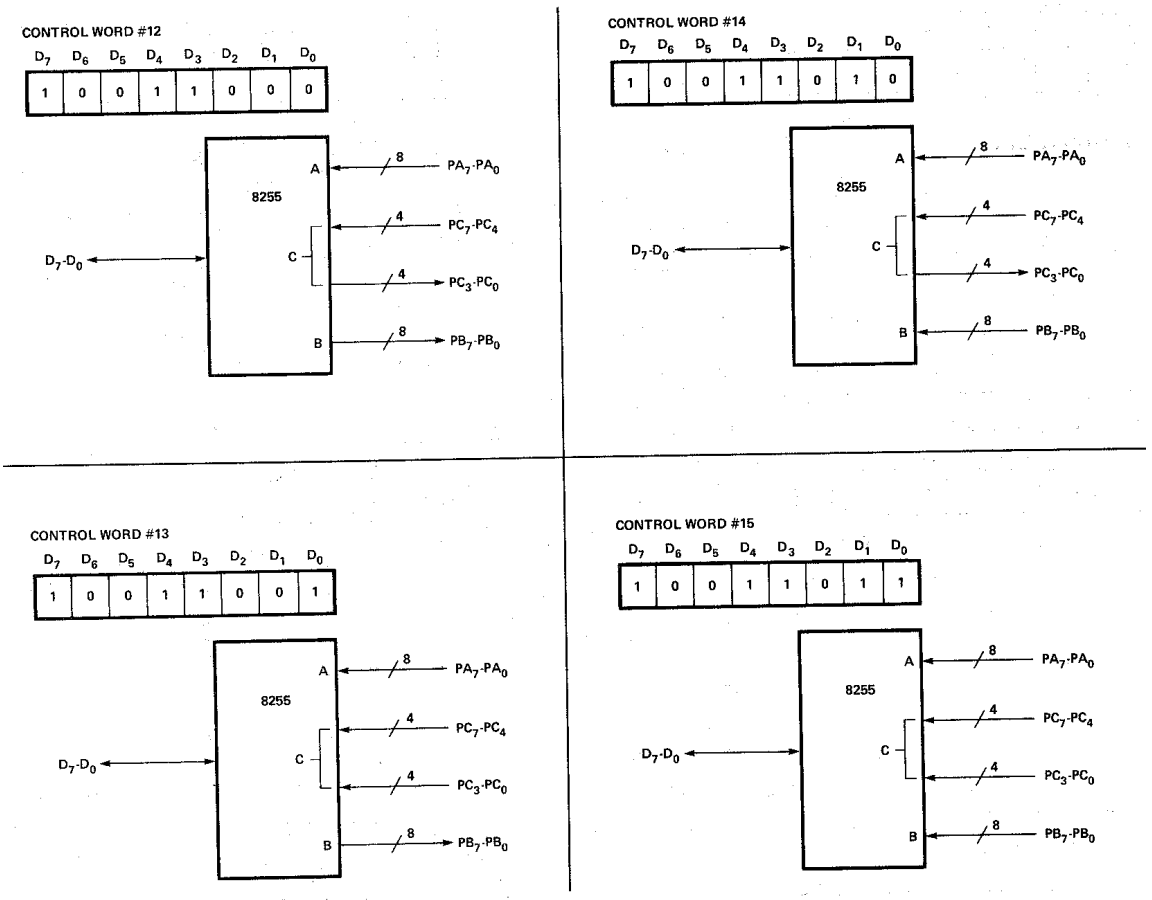


CONTROL WORD #11

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	1



5-119



Resumo dos bytes de programação da 8255 – PPI

80h	81h	82h	83h	88h	89h
8 |→ PA	8 |→ PA	8 |→ PA	8 |→ PA	8 |→ PA	8 |→ PA
2 |→ PC+	2 |→ PC+	2 |→ PC+	2 |→ PC+	2 |← PC+	2 |← PC+
5 |→ PC-	5 |← PC-	5 |→ PC-	5 |← PC-	5 |→ PC-	5 |← PC-
5 |→ PB	5 |→ PB	5 |← PB	5 |← PB	5 |→ PB	5 |→ PB

8Ah	8Bh	90h	91h	92h	93h
8 |→ PA	8 |→ PA	8 |← PA	8 |← PA	8 |← PA	8 |← PA
2 |← PC+	2 |← PC+	2 |→ PC+	2 |→ PC+	2 |→ PC+	2 |→ PC+
5 |→ PC-	5 |← PC-	5 |→ PC-	5 |← PC-	5 |→ PC-	5 |← PC-
5 |← PB	5 |← PB	5 |→ PB	5 |→ PB	5 |← PB	5 |← PB

98h	99h	9Ah	9Bh
8 |← PA	8 |← PA	8 |← PA	8 |← PA
2 |← PC+	2 |← PC+	2 |← PC+	2 |← PC+
5 |→ PC-	5 |← PC-	5 |→ PC-	5 |← PC-
5 |→ PB	5 |→ PB	5 |← PB	5 |← PB

O 74LS138:

O C.I. 74LS138 é um decodificador de três entradas para oito saídas, com três entradas de controle. Duas dessas três entradas são ativas em nível "0" (G2A' e G2B'), enquanto a terceira (G1) é ativa em nível "1".

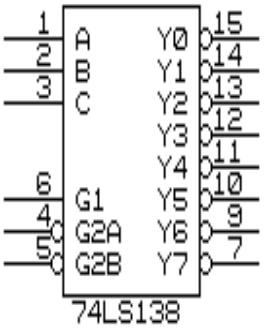
O funcionamento consiste em levar apenas uma das oito saídas para o nível "0", de acordo com a combinação das outras três entradas de seleção (A B C). Se pelo menos uma das entradas de controle estiver desabilitada, todas as saídas permanecerão em nível "1".

No circuito da interface, a saída Y0 é empregada para ativar a 8255 mostrada, porém as sete saídas restantes podem ser usadas para expansão do sistema isto é, para ativar outros periféricos conforme necessidade.

Tabela verdade do 74LS138:

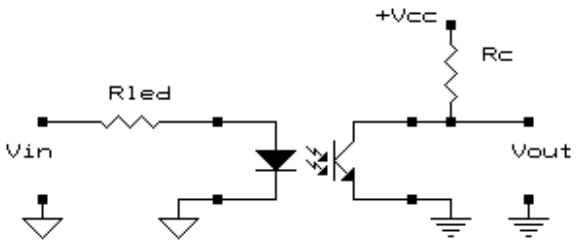
G1	G2A'	G2B'	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
0	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	1	X	X	X	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	0	0	1	1	1	1	0	1	1	1
1	0	0	1	0	1	1	1	1	1	1	0	1	1
1	0	0	1	1	0	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0

O símbolo do 74LS138:



O Foto-acoplador:

A figura seguinte representa um foto-acoplador (óptico-isolador, óptico-acoplador ou acoplador óptico) mostrando um LED acionando um fototransistor em um único encapsulamento. O funcionamento desta configuração, numa aplicação digital, consiste em acender ou apagar o LED de acordo com os níveis lógicos do sinal aplicado na entrada e assim, cortar ou saturar o fototransistor associado.



A forma de ligação desse componente deve ser adaptada a cada aplicação bem como, o valor do resistor ligado em série com o LED e do resistor ligado no coletor (ou emissor) do fototransistor escolhidos em função das tensões envolvidas.

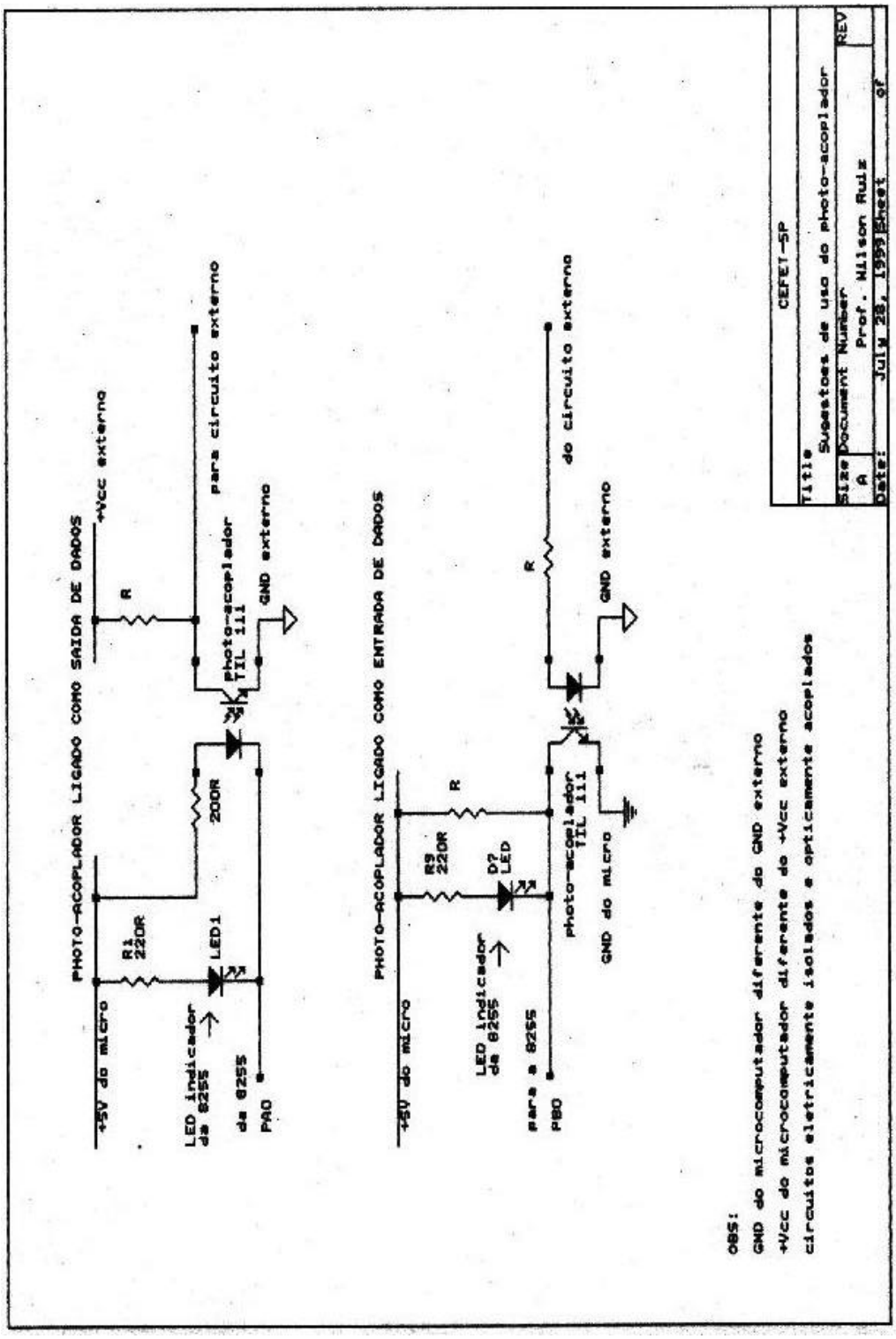
A grande vantagem do emprego de um foto-acoplador é a isolação elétrica entre os circuitos de entrada e saída (ou do computador e da etapa de potência da aplicação), pois separados os "terras" da entrada e da saída, não existe nenhum caminho condutivo entre os dois circuitos ficando estes então, eletricamente isolados e opticamente acoplados.

Existem vários tipos de foto-acopladores que, estruturalmente diferem no elemento de saída, podendo este ser um fotodiodo, um fototransistor, um fototransistor darlington ou um fototiristor, todos com características de isolação a partir de centenas de megaohms.

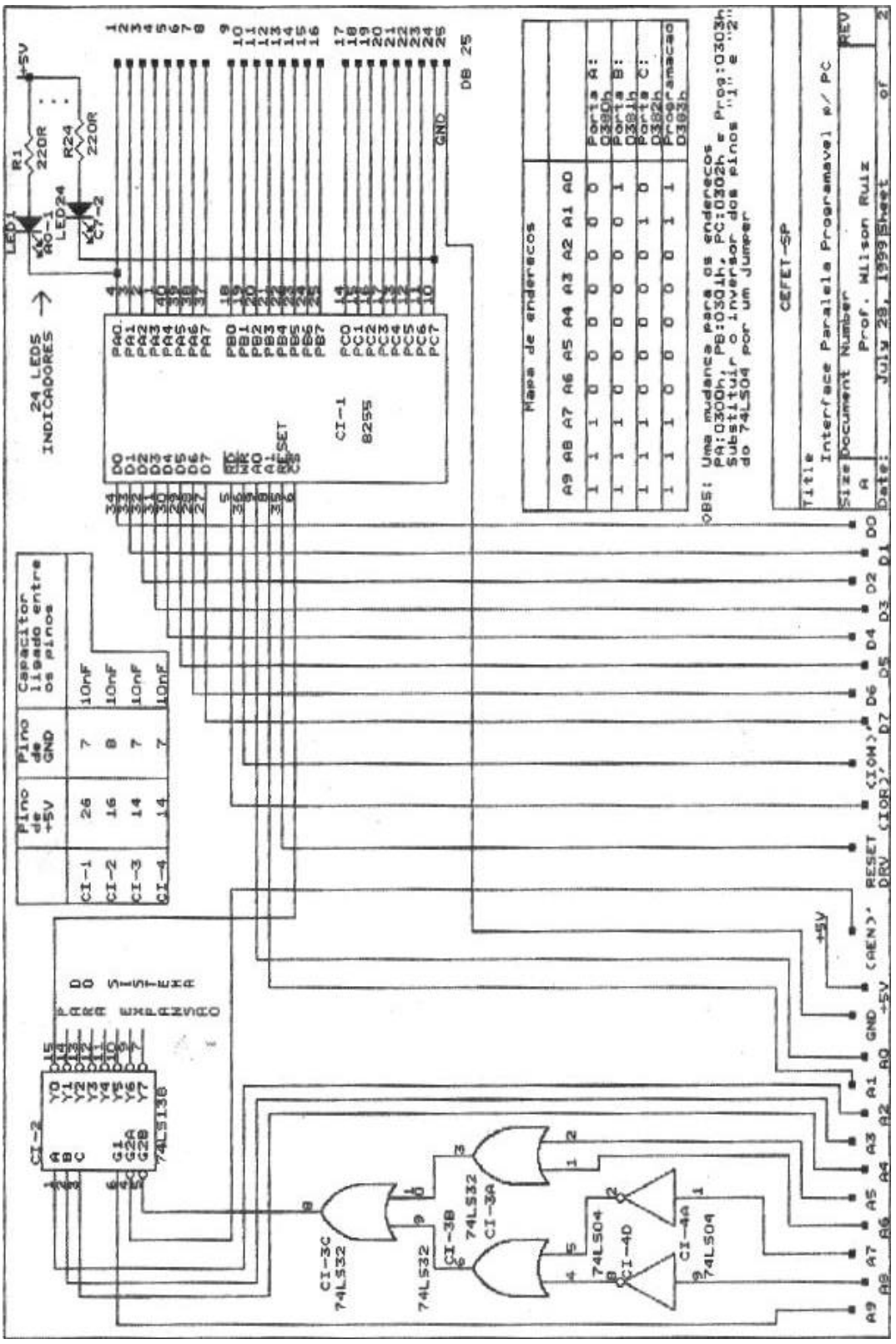
Circuitos:

Os esquemas dos circuitos sugeridos para a interface e para o uso dos foto-acopladores são mostrados a seguir.

SUGESTOES PARA O USO DOS FOTO-ACLOPADORES:



INTERFACE PARALELA PROGRAMÁVEL PARA PC



4.5 Normas para o projeto da disciplina:

Informações iniciais:

- Projeto realizado por um grupo constituído de preferencialmente 3 alunos.
- O trabalho tem como objetivo a elaboração completa de um "pequeno sistema controlado por um microcomputador".

Apresentação dividida em duas etapas:

- Placa de interface conectada a um slot ISA para PC funcionando.
- Utilização do circuito montado para o controle do sistema previamente definido pelo grupo.

Metodologia de trabalho:

- O grupo deve inicialmente cadastrar-se com o Professor.
- Esse grupo também deve apresentar propostas de sistemas ao Professor, até a data marcada e escolher, dentre as aprovadas, a que se constituirá no trabalho (estas propostas podem conter partes das sugestões apresentadas pelo Professor).
- Serão antecipadamente marcados "dias de desenvolvimento" onde os grupos deverão trazer seus respectivos projetos, para testes no laboratório (uma data para cada etapa descrita anteriormente).
- O exemplo de projeto de interface fornecido poderá ser usado integralmente ou adaptado para as necessidades do grupo.
- Outros circuitos envolvendo I/O's programáveis ou não, CI's TTL e/ou CMOS também serão aceitos.
- Para projetos que envolvam "sistemas de potência elevada ou acionamento eletromecânico" é obrigatório o uso de foto-acopladores, ligados como no esquema exemplo fornecido.
- São aceitos projetos com temas semelhantes porém, totalmente proibidos, projetos iguais (exceto na topologia usada para a interface).

Produto final esperado:

- O grupo deve apresentar a placa de interface, na forma de um protótipo, constituído por uma PCI com pente ISA (necessária para a conexão ao microcomputador) ligada, através de uma cabo, a outra tipo padrão e/ou montagem em "wire-wrap" e/ou "proto-board".
- O sistema controlado.
- Um manual do projeto contendo:
  - Diagrama de blocos completo.
  - Esquema eletrônico completo (que deverá incluir a interface apenas quando o projeto exemplo for alterado).
  - Explicação detalhada do funcionamento.
  - Listagem do software de controle (em Assembly) comentada.

Avaliação:

- As apresentações deverão ser feitas exclusivamente nas datas marcadas e na ordem definida anteriormente por sorteio (através do número do grupo).
- **A NOTA INDIVIDUAL**, será composta por fatores correspondentes a:

  -Efetiva participação do aluno nas datas marcadas para desenvolvimento.

  -Avaliação do funcionamento do Projeto.

  -Documentação entregue.

  -Argüição, feita pelo Professor, individualmente à cada aluno do grupo.

Observação:

- Eventuais situações não contempladas por este texto, serão julgadas pelo Professor da disciplina.

Calendário de desenvolvimento (para controle dos alunos):

- Cadastramento dos grupos de trabalho: _____
- Apresentação e análise da proposta de projeto: _____
- Data reservada para o desenvolvimento da interface: _____
- Apresentação da interface: _____
- Data reservada para o desenvolvimento do sistema:_____
- Apresentação final e entrega da documentação: _____

Grupo de trabalho:

Nome: **Nº:**

_____ _____

_____ _____

_____ _____

CAPITULO 5 – DIVERSOS

5.1 Exercícios de programação assembly
(para os microprocessadores Intel das famílias: 80X86, 80X88 e Pentium).

O objetivo desta lista (teórica) é apenas familiarizar o aluno com esta linguagem, enfatizando-se a aplicações das instruções básicas, de diferentes modos de endereçamento, más sem o rigor das definições dos mesmos e ainda, não importando-se com o ambiente de usado, assim ela deve ser resolvida seqüencialmente e logo no início da aprendizagem de programação.

1. Escrever um programa, em linguagem Assembly, dos microprocessadores relacionados, que realize as seguintes cópias de dados (conteúdos dos endereços de off-set):

Memória RAM (origem)			**Memória RAM (destino)**	
endereço	**dado**		**endereço**	**dado**
0150h	[ ]	→	0200h	[ ]
0151h	[ ]	→	0201h	[ ]
0152h	[ ]	→	0202h	[ ]
.			.	
.			.	
.			.	
017Ah	[ ]	→		

OBS1: Em todos os enunciados (nos exercícios seguintes) fica subentendido a seguinte frase: "Escrever um programa em linguagem Assembly, dos microprocessadores...".

OBS2: Por razões didáticas, sempre que possível, tomar como exemplo um programa já resolvido em um exercício anterior.

2. Idem ao anterior 1, somando o byte 05h, em cada conteúdo copiado.

3. Idem ao número 1, porém "pulando-se" 4 (quatro) endereços destino a cada cópia.

   (0150h) → (0200h) ; (0151h) → (0204h) ; ...

4. Idem ao número 1, realizando uma cópia inversa (invertendo os endereços da região de memória destino).

(0150h) → (022Ah) ; (0151h) → (0229h) ; ...

5. Idem ao número 1, subtraindo cada conteúdo, antes de ser copiado, do byte 0Eh.

6. Idem ao número 1, "pulando-se" 3 (três) endereços da memória origem, a cada conteúdo lido e mantendo-se o mesmo número total de bytes copiados.

   (0150h) → (0200h) ; (0153h) → (0201h) ; ...

7. Carregar (escrever) o byte 00h na seguinte região de memória RAM:

   de 0250h à 02BFh

   (0250h) ← 00h
   (0251h) ← 00h
   .
   .
   .
   (02BFh) ← 00h

8. Escrever 00h, 01h, 02h, 03h,..., na seguinte região de memória RAM:

   de 0200h à 02F9h

   (0200h) ← 00h
   (0201h) ← 01h
   .
   .
   .
   (02F9h) ← F9h

9. Fazer uma operação AND com o byte 55h e depois, com o resultado, uma nova operação OR com o byte 80h, para cada conteúdo da região de memória do exercício anterior e gravar cada resultado, na mesma posição de memória.

10. Contar o número de bytes iguais a 55h, que existem na seguinte região de memória:

    de 0185h à 030Bh

    Escrever o resultado final da contagem nos endereços 0183h e 0184h.

11. Idem ao anterior porém contando a seguinte seqüência de bytes:

    AAh, 55h (nessa ordem).

12. Pesquisar na seguinte região de memória (de 0200h à 027Fh), em ordem crescente de endereços e determinar em qual desses endereços encontra-se o primeiro byte 00h. Este endereço deve ficar armazenado no registrador BX, como resultado final. Caso não seja encontrado nenhum byte com esse valor o programa deve carregar FFFFh neste registrador.

5.2 Exemplo de um programa

Com o objetivo de ampliar as possibilidades de aplicações deste estudo, é fornecido a seguir a listagem fonte completa de um programa para o controle de um robô, escrito em assembly, no TASM (ANA14.ASM).

O hardware utilizado nessa aplicação é a placa de interface apresentada no capítulo 4.

O arquivo fonte está disponível aos alunos para estudos e testes.

```
;*****************************************************************************************************************
;  Projeto:              Controle do Robô ANA-2000  versão 1.4
;
;  Autor:                        Prof. Wilson Ruiz               data: 11/2000
;  Descrição:                    Braço mecânico, com 4 motores e um solenóide, controlado por PC, via placa de interface
;                                conectada em slot padrão ISA.
;  Operação:                     A cada uma das teclas abaixo atribui-se as respectivas funções:
;  tecla             Comando
;  (letras minúsculas)
;  M ..... Inicia modo manual de controle
;  A .....  Movimenta M0 no sentido horário ou para M0
;  S .....  Movimenta M0 no sentido anti-horário ou para M0
;  D .....  Movimenta M1 no sentido horário ou para M1
;  F .....  Movimenta M1 no sentido anti-horário ou para M1
;  G ..... Movimenta M2 no sentido horário ou para M2
;  H ..... Movimenta M2 no sentido anti-horário ou para M2
;  J .....  Movimenta M3 no sentido horário ou para M3
;  K ..... Movimenta M3 no sentido anti-horário ou para M3
;  L .....  Abre / fecha garra
;  X ..... Desliga todos os motores e abre a garra
;  I .....  Informa as teclas de comandos
;  Z .....  Sai do programa
;  1 .....  Seleciona a velocidade 1 (mais lenta)
;  2 .....  Seleciona a velocidade 2
;  3 .....  Seleciona a velocidade 3
;  4 .....  Seleciona a velocidade 4 (mais rápida)
;  ligação dos motores:                                  Bobina      Bits da
;                                                                    8255
;  Motor 0 = M0 (base) .....                             A0 ..... PA0
;                                                        A0'..... PA1
;                                                        B0 ..... PA2
;                                                        B0'..... PA3
;  Motor 1 = M1  (ombro).....                            A1 ..... PA4
;                                                        A1'..... PA5
;                                                        B1 ..... PA6
;                                                        B1'..... PA7
;  Motor 2 = M2 (cotovelo) ....                          A2 ..... PB0
;                                                        A2'..... PB1
;                                                        B2 ..... PB2
;                                                        B2'..... PB3
;  Motor 3 = M3 (punho) .....                            A3 ..... PB4
;                                                        A3'..... PB5
;                                                        B3 ..... PB6
;                                                        B3'..... PB7
; LIGACAO DOS SENSORES:
; nome                          função                  bits da 8255
; S0_MO..... sensor de fim de curso p/ M0 (zero) .....  PC0
; S1_M0   ..... sensor de fim de curso p/ M0 .....      PC1
; S0_M1   ..... sensor de fim de curso p/ M0 (zero)..... PC2
; S1_M1   ..... sensor de fim de curso p/ M0 .....      PC3
; S0_M2   ..... sensor de fim de curso p/ M0 (zero) ..... PC3
; S1_M2   ..... sensor de fim de curso p/ M0 .....      PC5
; S0_M3   ..... sensor de fim de curso p/ M0 (zero) ..... PC6
; S1_M3   ..... sensor de fim de curso p/ M0 .....      PC7
;  ligação da garra:
;                                                        bits do
;                                                        74LS373
;  solenóide .....                                       D0
;           Programas:
;           editor de texto:                             NE
;           Compilador:                                  TASM
;           Liker:                                       TLINK
;*****************************************************************************************************************
;nome do programa: ana14.asm

CR      EQU     13                      ;ASCII Carriage Return
LF      EQU     10                      ;ASCII Line Feed
        DOSSEG
        .MODEL small
        .STACK
```

```
        .DATA
;*****************************************************************************************************************
;       REGIAO DE DEFINICAO DE VARIAVEIS
;*****************************************************************************************************************
BYTE_M0                 DB      11H             ;para acionamento inicial do motor 0
BYTE_M1                 DB      11H             ;para acionamento inicial do motor 1
BYTE_M2                 DB      0EEH            ;para acionamento inicial do motor 2
BYTE_M3                 DB      0EEH            ;para acionamento inicial do motor 3
MIRROR_M0               DB      11H             ;byte espelho do motor 0
MIRROR_M1               DB      11H             ;byte espelho do motor 1
MIRROR_M2               DB      0EEH            ;byte espelho do motor 2
MIRROR_M3               DB      0EEH            ;byte espelho do motor 3
BYTE_VEL                DB      0FFH            ;byte de seleção da velocidade
                                                ;do motor escolhido
MIRROR_GARRA   DB       00H                     ;byte do status da garra
;*****************************************************************************************************************
;       REGIAO DE DEFINICAO DE CONSTANTES
;*****************************************************************************************************************
PA                      DW      0380H           ;endereço da porta A da 8255
PB                      DW      0381H           ;endereço da porta B da 8255
PC                      DW      0382H           ;endereço da porta C da 8255
PROG                    DW      0383H           ;endereço de programação da 8255
GARRA                   DW      0384H           ;endereço de acesso a garra
MENS_M0_H               DB  cr,lf,lf,' -> Movimentando motor 0 no sentido horario$'
MENS_M0_AH              DB  cr,lf,lf,' -> Movimentando motor 0 no sentido anti-horario$'
MENS_M1_H               DB  cr,lf,lf,' -> Movimentando motor 1 no sentido horario$'
MENS_M1_AH              DB  cr,lf,lf,' -> Movimentando motor 1 no sentido anti-horario$'
MENS_M2_H               DB  cr,lf,lf,' -> Movimentando motor 2 no sentido horario$'
MENS_M2_AH              DB  cr,lf,lf,' -> Movimentando motor 2 no sentido anti-horario$'
MENS_M3_H               DB  cr,lf,lf,' -> Movimentando motor 3 no sentido horario$'
MENS_M3_AH              DB  cr,lf,lf,' -> Movimentando motor 3 no sentido anti-horario$'
MENS_M0_STOPED          DB  cr,lf,lf,' -> Motor 0 parado$'
MENS_M1_STOPED          DB  cr,lf,lf,' -> Motor 1 parado$'
MENS_M2_STOPED          DB  cr,lf,lf,' -> Motor 2 parado$'
MENS_M3_STOPED          DB  cr,lf,lf,' -> Motor 3 parado$'
MENS_V1                 DB  cr,lf,lf,' # Velocidade 1 selecionada #$'
MENS_V2                 DB  cr,lf,lf,' # Velocidade 2 selecionada #$'
MENS_V3                 DB  cr,lf,lf,' # Velocidade 3 selecionada #$'
MENS_V4                 DB  cr,lf,lf,' # Velocidade 4 selecionada #$'
MENS_G_OPENED           DB  cr,lf,lf,' * Garra aberta *$'
MENS_G_CLOSED           DB  cr,lf,lf,' * Garra fechada *$'
MENS_INICIAL            DB  cr,lf,lf,' > > PROGRAMA DE CONTROLE DO ROBO ANA - 2000 < <$'
MENS_M_OFF              DB  cr,lf,lf,' >>>> Todos os motores desligados <<<$'
MENS_FINAL              DB  cr,lf,lf,lf,lf,lf,' ------ Programa finalizado ------'
                        DB  cr,lf,lf,lf,'  O ANA - 2000 despede-se: TCHAU!$'
MENS_MENU               DB  cr,lf,lf,lf,lf,' MENU INICIAL - tecle a letra minuscula'
                        DB  cr,lf,lf,' M .............Inicia o modo manual de controle'
                        DB  cr,lf,lf,' I .............Informa teclas de comando$'
MENS_M_MANUAL           DB  cr,lf,lf,lf,lf,' Selecionado o MODO DE CONTROLE MANUAL!$'
MENS_TECLAS             DB  cr,lf,lf,lf,lf,' TECLAS DE COMANDOS E SUAS FUNCOES:'
                        DB  cr,lf,lf,' TECLAS:                  COMANDOS:'
                        DB  cr,lf,' (letras'
                        DB CR,LF,' minusculas'
                        DB cr,lf,' ou numeros)'
 DB cr,lf,'  M ---> Inicia modo manual de controle'
 DB cr,lf,'  A ---> Movimenta M0 no sentido horario ou para M0.....(BASE)'
 DB cr,lf,'  S ---> Movimenta M0 no sentido anti-horario ou para M0.(BASE)'
 DB cr,lf,'  D ---> Movimenta M1 no sentido horario ou para M1.....(OMBRO)'
 DB cr,lf,'  F ---> Movimenta M1 no sentido anti-horario ou para M1.(OMBRO)'
 DB cr,lf,'  G ---> Movimenta M2 no sentido horario ou para M2.....(COTOVELO)'
 DB cr,lf,'  H ---> Movimenta M2 no sentido anti-horario ou para M2.(COTOVELO)'
 DB cr,lf,'  J ---> Movimenta M3 no sentido horario ou para M3.....(PUNHO)'
 DB cr,lf,'  K ---> Movimenta M3 no sentido anti-horario ou para M3.(PUNHO)'
 DB cr,lf,'  L ---> Abre / fecha garra'
 DB cr,lf,'  X ---> Desliga todos os motores e abre a garra'
 DB cr,lf,'  I ---> Informa as teclas de comandos'
 DB cr,lf,'  Z ---> Sai do programa'
 DB cr,lf,'  1 ---> Seleciona a velocidade 1.............(mais lenta)'
 DB cr,lf,'  2 ---> Seleciona a velocidade 2'
 DB cr,lf,'  3 ---> Seleciona a velocidade 3'
```

```
 DB cr,lf,'  4 ---> Seleciona a velocidade 4.............(mais rapida)$'

        .CODE
ANA1    PROC                                ;necessário para enviar mensagens para a tela
        mov     ax, @data                   ;inicializa os registradores de
        mov     ds, ax                      ;segmento DS e ES
        mov     es, ax
        cld                                 ;Clear Direction Flag
        push    bp                          ;moldura
        mov     bp, sp                      ;salva ponteiro de base anterior
        mov     ah, 0BH                     ;servico 0BH, define cor
        mov     bh, 00H                     ;define fundo
        mov     bl, 03H                     ;usa cor 01 (azul)
        int     10H                         ;requisita o serviço de vídeo
        pop     bp                          ;recupera o ponteiro de base anterior
        mov     dx, PROG                    ;programa 8255
        mov     al, 89H                     ;PA, PB e PC como saídas
        out     dx, al
        mov     dx, PA                      ;estabelece status inicial dos
        mov     al, 11H                     ;motores, necessária alteração
        out     dx, al                      ;em função dos drives escolhidos
        mov     dx, PB                      ;para cada motor
        mov     al, 0EEH
        out     dx, al
        call    ESCREVE_INICIO              ;escreve a mensagem inicial
        call    ESCREVE_MENU                ;escreve o menu inicial
L18:    mov     ah, 08H                     ;leitura de teclado sem eco
        int     21H
        cmp     al, 6DH                     ;verifica tecla "M" minúscula
        jnz     L17
        call    ESCREVE_M_MANUAL
        jmp     L16
L17:    cmp     al, 69H                     ;verifica tecla "I" minúscula
        jnz     L18
        call    ESCREVE_TECLAS              ;escreve lista de teclas de comandos
        jmp     L18
L16:    mov     ah, 08H                     ;leitur a de teclado sem eco
        int     21H
        cmp     al, 31H                     ;verifica tecla "1"
        jnz     L1
        call    VEL_1                       ;seleciona velocidade 1
L1:     cmp     al, 32H                     ;verifica tecla "2"
        jnz     L2
        call    VEL_2                       ;seleciona velocidade 2
L2:     cmp     al, 33H                     ;verifica tecla "3"
        jnz     L3
        call    VEL_3                       ;seleciona velocidade 3
L3:     cmp     al, 34H                     ;verifica tecla "4"
        jnz     L4
        call    VEL_4                       ;seleciona velocidade 4
L4:     cmp     al, 61H                     ;verifica tecla "A" minúscula
        jnz     L5
        call    MOVE_M0_H                   ;movimenta o motor 0, sentido horário
L5:     cmp     al, 73H                     ;verifica tecla "S" minúscula
        jnz     L6
        call    MOVE_M0_AH                  ;movimenta o motor 0, sentido anti-horário
L6:     cmp     al, 64H                     ;verifica tecla "D" minúscula
        jnz     L7
        call    MOVE_M1_H                   ;movimenta o motor 1, sentido horário
L7:     cmp     al, 66H                     ;verifica tecla "F" minúscula
        jnz     L8
        call    MOVE_M1_AH                  ;movimenta o motor 1, sentido anti-horário
L8:     cmp     al, 67H                     ;verifica tecla "G" minúscula
        jnz     L9
        call    MOVE_M2_H                   ;movimenta o motor 2, sentido horário
L9:     cmp     al, 68H                     ;verifica tecla "H" minúsc ula
        jnz     L10
        call    MOVE_M2_AH                  ;movimenta o motor 2, sentido anti-horário
L10:    cmp     al, 6AH                     ;verifica tecla "J" minúscula
        jnz     L11
```

```
        call    MOVE_M3_H               ;movimenta o motor 3, sentido horário
L11:    cmp     al, 6BH                 ;verifica tecla "K" minúscula
        jnz     L12
        call    MOVE_M3_AH              ;movimenta o motor 3, sentido anti-horário
L12:    cmp     al, 6CH                 ;verifica tecla "L" minúscula
        jnz     L13
        call    COMUTA_GARRA            ;comuta a solenóide da garra
L13:    cmp     al, 69H                 ;verifica tecla "I" minúscula
        jnz     L14
        call    ESCREVE_TECLAS          ;escreve lista de teclas de comandos
L14:    cmp     al, 7AH                 ;verifica tecla "Z" minúscula
        jz      L15
        jmp     L16                     ;retorna para ler teclado
L15:    call    ESCREVE_FINAL           ;escreve mensagem final
        mov     ah, 4CH                 ;retorna ao sistema operacional
        int     21H
ANA1    ENDP                            ;finaliza o procedimento

;*************************************************************************************************************
;Subrotina:      MOVE_M0_H
;               Lê o byte espelho de movimento do motor 0; MIRROR_M0
;               Move o motor 0 no sentido horário. Atualiza MIRROR_M0
;               Continua o movimento do motor ate que a tecla correspondente seja acionada pela segunda vez
;               Aciona a escrita das mensagens correspondentes ao status do motor selecionado
;*************************************************************************************************************
PUBLIC MOVE_M0_H
MOVE_M0_H       PROC
        push    ax
        push    dx
        call    ESCREVE_M0_H            ;mensagem
M1:     mov     dx, PA
        mov     ah, MIRROR_M0           ;lê byte espelho
        rol     ah, 1                   ;desloca
        mov     MIRROR_M0, ah           ;atualiza byte espelho
        and     ah, 0FH
        in      al, dx                  ;lê a porta A
        and     al, 0F0H
        or      al, ah                  ;mascara
        out     dx, al                  ;move M0 de um passo
        call    PAUSA_1                 ;pausa
        mov     ah, 0BH
        int     21H                     ;lê buffer do teclado
        cmp     al, 0FFH                ;verifica se existe tecla
        jnz     M1                      ;se não existe tecla continua movimento
        mov     ah, 07H
        int     21H                     ;lê teclado
        cmp     al, 61H                 ;compara com a tecla "A" minúscula
        jnz     M1                      ;se diferente continua movimento
        call    ESCREVE_M0_STOP         ;mensagem
        pop     dx
        pop     ax
        ret
MOVE_M0_H       ENDP                    ;finaliza o procedimento

;*************************************************************************************************************
;Subrotina:      MOVE_M0_AH
;               Lê o byte espelho de movimento do motor 0; MIRROR_M0
;               Move o motor 0 no sentido anti-horário. Atualiza MIRROR_M0
;               Continua o movimento do motor ate que a tecla correspondente seja acionada pela segunda vez
;               Aciona a escrita das mensagens correspondentes ao status do motor selecionado
;*************************************************************************************************************
PUBLIC MOVE_M0_AH
MOVE_M0_AH      PROC
        push    ax
        push    dx
        call    ESCREVE_M0_AH           ;mensagem
M2:     mov     dx, PA
        mov     ah, MIRROR_M0           ;lê byte espelho
        ror     ah, 1                   ;desloca
        mov     MIRROR_M0, ah           ;atualiza byte espelho
```

```
        and     ah, 0FH
        in      al, dx                  ;lê a porta A
        and     al, 0F0H
        or      al, ah                  ;mascara
        out     dx, al                  ;move M0 de um passo
        call    PAUSA_1                 ;pausa
        mov     ah, 0BH
        int     21H                     ;lê buffer do teclado
        cmp     al, 0FFH                ;verifica se existe tecla
        jnz     M2                      ;se não existe tecla continua movimento
        mov     ah, 07H
        int     21H                     ;lê teclado
        cmp     al, 73H                 ;compara com a tecla "S" minúscula
        jnz     M2                      ;se diferente continua movimento
        call    ESCREVE_M0_STOP         ;mensagem
        pop     dx
        pop     ax
        ret
MOVE_M0_AH  ENDP                        ;finaliza o procedimento

;*************************************************************************************************************
;Subrotina:      MOVE_M1_H
;               Lê o byte espelho de movimento do motor 1; MIRROR_M1
;               Move o motor 1 no sentido horário. Atualiza MIRROR_M1
;               Continua o movimento do motor ate que a tecla correspondente seja acionada pela segunda vez
;               Aciona a escrita das mensagens correspondentes ao status do motor selecionado
;*************************************************************************************************************
PUBLIC MOVE_M1_H
MOVE_M1_H   PROC
        push    ax
        push    dx
        call    ESCREVE_M1_H            ;mensagem
M3:     mov     dx, PA
        mov     ah, MIRROR_M1           ;lê byte espelho
        rol     ah, 1                   ;desloca
        mov     MIRROR_M1, ah           ;atualiza byte espelho
        and     ah, 0 F0H
        in      al, dx                  ;lê a porta A
        and     al, 0FH
        or      al, ah                  ;mascara
        out     dx, al                  ;move M1 de um passo
        call    PAUSA_1                 ;pausa
        mov     ah, 0BH
        int     21H                     ;lê buffer do teclado
        cmp     al, 0FFH                ;verifica se existe tecla
        jnz     M3                      ;se não existe tecla continua movimento
        mov     ah, 07H
        int     21H                     ;lê teclado
        cmp     al, 64H                 ;compara com a tecla "D" minúscula
        jnz     M3                      ;se diferente continua movimento
        call    ESCREVE_M1_STOP         ;mensagem
        pop     dx
        pop     ax
        ret
MOVE_M1_H   ENDP                        ;finaliza o procedimento

;*************************************************************************************************************
;Subrotina:      MOVE_M1_AH
;               Lê o byte espelho de movimento do motor 1; MIRROR_M1
;               Move o motor 1 no sentido anti-horário. Atualiza MIRROR_M1
;               Continua o movimento do motor ate que a tecla correspondente seja acionada pela segunda vez
;               Aciona a escrita das mensagens correspondentes ao status do motor selecionado
;*************************************************************************************************************
PUBLIC MOVE_M1_AH
MOVE_M1_AH  PROC
        push    ax
        push    dx
        call    ESCREVE_M1_AH           ;mensagem
M4:     mov     dx, PA
        mov     ah, MIRROR_M1           ;lê byte espelho
```

```
        ror     ah, 1                   ;desloca
        mov     MIRROR_M1, ah           ;atualiza byte espelho
        and     ah, 0F0H
        in      al, dx                  ;lê a porta A
        and     al, 0FH
        or      al, ah                  ;mascara
        out     dx, al                  ;move M1 de um passo
        call    PAUSA_1                 ;pausa
        mov     ah, 0BH
        int     21H                     ;lê buffer do teclado
        cmp     al, 0FFH                ;verifica se existe tecla
        jnz     M4                      ;se não existe tecla continua movimento
        mov     ah, 07H
        int     21H                     ;lê teclado
        cmp     al, 66H                 ;compara com a tecla "F" minúscula
        jnz     M4                      ;se diferente continua movimento
        call    ESCREVE_M1_STOP         ;mensagem
        pop     dx
        pop     ax
        ret
MOVE_M1_AH  ENDP                        ;finaliza o procedimento

;*************************************************************************************************************
;Subrotina:      MOVE_M2_H
;               Lê o byte espelho de movimento do motor 2; MIRROR_M2
;               Move o motor 2 no sentido horário. Atualiza MIRROR_M2
;               Continua o movimento do motor ate que a tecla correspondente seja acionada pela segunda vez
;               Aciona a escrita das mensagens correspondentes ao status do motor selecionado
;*************************************************************************************************************
PUBLIC MOVE_M2_H
MOVE_M2_H   PROC
        push    ax
        push    dx
        call    ESCREVE_M2_H            ;mensagem
M5:     mov     dx, PB
        mov     ah, MIRROR_M2           ;lê byte espelho
        rol     ah, 1                   ;desloca
        mov     MIRROR_M2, ah           ;atualiza byte espelho
        and     ah, 0FH
        in      al, dx                  ;lê a porta B
        and     al, 0F0H
        or      al, ah                  ;mascara
        out     dx, al                  ;move M2 de um passo
        call    PAUSA_1                 ;pausa
        mov     ah, 0BH
        int     21H                     ;lê buffer do teclado
        cmp     al, 0FFH                ;verifica se existe tecla
        jnz     M5                      ;se não existe tecla continua movimento
        mov     ah, 07H
        int     21H                     ;lê teclado
        cmp     al, 67H                 ;compara com a tecla "G" minúscula
        jnz     M5                      ;se diferente continua movimento
        call    ESCREVE_M2_STOP         ;mensagem
        pop     dx
        pop     ax
        ret
MOVE_M2_H   ENDP                        ;finaliza o procedimento

;*************************************************************************************************************
;Subrotina:      MOVE_M2_AH
;               Lê o byte espelho de movimento do motor 2; MIRROR_M2
;               Move o motor 2 no sentido anti-horário. Atualiza MIRROR_M2
;               Continua o movimento do motor ate que a tecla correspondente seja acionada pela segunda vez
;               Aciona a escrita das mensagens correspondentes ao status do motor selecionado
;*************************************************************************************************************
PUBLIC MOVE_M2_AH
MOVE_M2_AH  PROC
        push    ax
        push    dx
        call    ESCREVE_M2_AH           ;mensagem
```

```
M6:     mov     dx, PB
        mov     ah, MIRROR_M2           ;lê byte espelho
        ror     ah, 1                   ;desloca
        mov     MIRROR_M2, ah           ;atualiza byte espelho
        and     ah, 0FH
        in      al, dx                  ;lê a porta B
        and     al, 0F0H
        or      al, ah                  ;mascara
        out     dx, al                  ;move M2 de um passo
        call    PAUSA_1                 ;pausa
        mov     ah, 0BH
        int     21H                     ;lê buffer do teclado
        cmp     al, 0FFH                ;verifica se existe tecla
        jnz     M6                      ;se não existe tecla continua movimento
        mov     ah, 07H
        int     21H                     ;lê teclado
        cmp     al, 68H                 ;compara com a tecla "H" minúscula
        jnz     M6                      ;se diferente continua movimento
        call    ESCREVE_M2_STOP         ;mensagem
        pop     dx
        pop     ax
        ret
MOVE_M2_AH  ENDP                        ;finaliza o procedimento

;*************************************************************************************************************
;Subrotina:     MOVE_M3_H
;               Lê o byte espelho de movimento do motor 3; MIRROR_M3
;               Move o motor 3 no sentido horário. Atualiza MIRROR_M3
;               Continua o movimento do motor ate que a tecla correspondente seja acionada pela segunda vez
;               Aciona a escrita das mensagens correspondentes ao status do motor selecionado
;*************************************************************************************************************
PUBLIC MOVE_M3_H
MOVE_M3_H   PROC
        push    ax
        push    dx
        call    ESCREVE_M3_H            ;mensagem
M7:     mov     dx, PB
        mov     ah, MIRROR_M3           ;lê byte espelho
        rol     ah, 1                   ;desloca
        mov     MIRROR_M3, ah           ;atualiza byte espelho
        and     ah, 0F0H
        in      al, dx                  ;lê a porta B
        and     al, 0FH
        or      al, ah                  ;mascara
        out     dx, al                  ;move M3 de um passo
        call    PAUSA_1                 ;pausa
        mov     ah, 0BH
        int     21H                     ;lê buffer do teclado
        cmp     al, 0FFH                ;verifica se existe tecla
        jnz     M7                      ;se não existe tecla continua movimento
        mov     ah, 07H
        int     21H                     ;lê teclado
        cmp     al, 6AH                 ;compara com a tecla "J" minúscula
        jnz     M7                      ;se diferente continua movimento
        call    ESCREVE_M3_STOP         ;mensagem
        pop     dx
        pop     ax
        ret
MOVE_M3_H   ENDP                        ;finaliza o procedimento

;*************************************************************************************************************
;Subrotina:     MOVE_M3_AH
;               Lê o byte espelho de movimento do motor 3; MIRROR_M3
;               Move o motor 3 no sentido anti-horário. Atualiza MIRROR_M3
;               Continua o movimento do motor ate que a tecla correspondente seja acionada pela segunda vez
;               Aciona a escrita das mensagens correspondentes ao status do motor selecionado
;*************************************************************************************************************
PUBLIC MOVE_M3_AH
MOVE_M3_AH  PROC
        push    ax
```

```
        push    dx
        call    ESCREVE_M3_AH           ;mensagem
M8:     mov     dx, PB
        mov     ah, MIRROR_M3           ;lê byte espelho
        ror     ah, 1                   ;desloca
        mov     MIRROR_M3, ah           ;atualiza byte espelho
        and     ah, 0F0H
        in      al, dx                  ;lê a porta B
        and     al, 0FH
        or      al, ah                  ;mascara
        out     dx, al                  ;move M3 de um passo
        call    PAUSA_1                 ;pausa
        mov     ah, 0BH
        int     21H                     ;lê buffer do teclado
        cmp     al, 0FFH                ;verifica se existe tecla
        jnz     M8                      ;se não existe tecla continua movimento
        mov     ah, 07H
        int     21H                     ;lê teclado
        cmp     al, 6BH                 ;compara com a tecla "K" minúscula
        jnz     M8                      ;se diferente continua movimento
        call    ESCREVE_M3_STOP         ;mensagem
        pop     dx
        pop     ax
        ret
MOVE_M3_AH  ENDP                        ;finaliza o procedimento

;*************************************************************************************************************
;Subrotina:      VEL_1
;               Atualiza o byte de velocidade para a velocidade numero 1. Escreve a mensagem correspondente
;*************************************************************************************************************
PUBLIC VEL_1
VEL_1           PROC
        push    ax
        push    dx
        push    bx                      ;necessário pelo caractere colorido
        push    cx                      ;necessário pelo caractere colorido
        mov     BYTE_VEL, 0FFH          ;atualiza byte de velocidade
        mov     ah, 09H
        lea     dx, MENS_V1
        int     21H                     ;mensagem
        push    bp                      ;cria caractere colorido
        mov     ah, 09H                 ;no final da mensagem
        mov     al, 16H
        mov     bl, 83H                 ;define atributo, cor = azul claro piscando
        mov     cx, 0003H
        int     10H
        pop     bp
        pop     cx
        pop     bx
        pop     dx
        pop     ax
        ret
VEL_1           ENDP                    ;finaliza o procedimento

;*************************************************************************************************************
;Subrotina:      VEL_2
;               Atualiza o byte de velocidade para a velocidade numero 2. Escreve a mensagem correspondente
;*************************************************************************************************************
PUBLIC VEL_2
VEL_2           PROC
        push    ax
        push    dx
        push    bx                      ;necessário pelo caractere colorido
        push    cx                      ;necessário pelo caractere colorido
        mov     BYTE_VEL, 7FH           ;atualiza byte de velocidade
        mov     ah, 09H
        lea     dx, MENS_V2
        int     21H                     ;mensagem
        push    bp                      ;cria caractere colorido
        mov     ah, 09H                 ;no final da mensagem
```

```
        mov     al, 16H
        mov     bl, 83H                 ;define atributo, cor = azul claro piscando
        mov     cx, 0006H
        int     10H
        pop     bp
        pop     cx
        pop     bx
        pop     dx
        pop     ax
        ret
VEL_2           ENDP                    ;finaliza o procedimento

;*********************************************************************************************************************
;Subrotina:     VEL_3
;               Atualiza o byte de velocidade para a velocidade numero 3. Escreve a mensagem correspondente
;*********************************************************************************************************************
PUBLIC VEL_3
VEL_3           PROC
        push    ax
        push    dx
        push    bx                      ;necessário pelo caractere colorido
        push    cx                      ;necessário pelo caractere colorido
        mov     BYTE_VEL, 3FH           ;atualiza byte de velocidade
        mov     ah, 09H
        lea     dx, MENS_V3
        int     21H                     ;mensagem
        push    bp                      ;cria caractere colorido
        mov     ah, 09H                 ;no final da mensagem
        mov     al, 16H
        mov     bl, 83H                 ;define atributo, cor = azul claro piscando
        mov     cx, 0009H
        int     10H
        pop     bp
        pop     cx
        pop     bx
        pop     dx
        pop     ax
        ret
VEL_3           ENDP                    ;finaliza o procedimento

;*********************************************************************************************************************
;Subrotina:     VEL_4
;               Atualiza o byte de velocidade para a velocidade numero 4. Escreve a mensagem correspondente
;*********************************************************************************************************************
PUBLIC VEL_4
VEL_4           PROC
        push    ax
        push    dx
        push    bx                      ;necessário pelo caractere colorido
        push    cx                      ;necessário pelo caractere colorido
        mov     BYTE_VEL, 07H           ;atualiza byte de velocidade
        mov     ah, 09H
        lea     dx, MENS_V4
        int     21H                     ;mensagem
        push    bp                      ;cria caractere colorido
        mov     ah, 09H                 ;no final da mensagem
        mov     al, 16H
        mov     bl, 83H                 ;define atributo, cor = azul claro piscando
        mov     cx, 000CH
        int     10H
        pop     bp
        pop     cx
        pop     bx
        pop     dx
        pop     ax
        ret
VEL_4           ENDP                    ;finaliza o procedimento

;*********************************************************************************************************************
;Subrotina:     COMUTA_GARRA
```

```
;                       Lê o byte de status da garra: MIRROR_GARRA. Comuta o status da solenóide. Atualiza o MIRROR_GARRA
;*****************************************************************************************************************
PUBLIC GARRA
COMUTA_GARRA  PROC
        push    ax
        push    dx
        mov     ah, MIRROR_GARRA
        cmp     ah, 00H
        jz      G1
        mov     dx, GARRA
        mov     al, 00H
        mov     MIRROR_GARRA, al
        out     dx, al
        call    ESCREVE_G_ABERTA
        jmp     G2
G1:     mov     dx, GARRA
        mov     al, 01H
        mov     MIRROR_GARRA, al
        out     dx, al
        call    ESCREVE_G_FECHADA
G2:     pop     dx
        pop     ax
        ret
COMUTA_GARRA  ENDP

;*****************************************************************************************************************
;Subrotina:             PAUSA_1
;*****************************************************************************************************************
PUBLIC PAUSA_1
PAUSA_1         PROC
        push    bx
        push    cx
        push    dx
        mov     bl, BYTE_VEL                    ;lê byte de movimento atual
P3:     mov     cx, 00ffH
P2:     mov     dx, 00ffH
P1:     dec     dx
        jnz     P1
        dec     cx
        jnz     P2
        dec     bl
        jnz     P3
        pop     dx
        pop     cx
        pop     bx
        ret
PAUSA_1         ENDP

;*****************************************************************************************************************
;Subrotina:             ESCREVE_INICIO
;                       Escreve a mensagem de apresentação inicial
;*****************************************************************************************************************
PUBLIC ESCREVE_INICIO
ESCREVE_INICIO  PROC
        push    ax
        push    dx
        mov     ah, 09H
        lea     dx, MENS_INICIAL
        int     21H                             ;mensagem
        pop     dx
        pop     ax
        ret
ESCREVE_INICIO  ENDP

;*****************************************************************************************************************
;Subrotina:             ESCREVE_M_MANUAL
;                       Escreve a mensagem de inicio do modo manual
;*****************************************************************************************************************
PUBLIC ESCREVE_M_MANUAL
ESCREVE_M_MANUAL PROC
```

```
        push    ax
        push    dx
        mov     ah, 09H
        lea     dx, MENS_M_MANUAL
        int     21H                             ;mensagem
        pop     dx
        pop     ax
        ret
ESCREVE_M_MANUAL ENDP

;*****************************************************************************************************************
;Subrotina:      ESCREVE_TECLAS
;               Escreve a lista de teclas de comandos
;*****************************************************************************************************************
PUBLIC ESCREVE_TECLAS
ESCREVE_TECLAS PROC
        push    ax
        push    dx
        mov     ah, 09H
        lea     dx, MENS_TECLAS
        int     21H                             ;mensagem
        pop     dx
        pop     ax
        ret
ESCREVE_TECLAS ENDP

;*****************************************************************************************************************
;Subrotina:      ESCREVE_MENU
;               Escreve o menu inicial
;*****************************************************************************************************************
PUBLIC ESCREVE_MENU
ESCREVE_MENU  PROC
        push    ax
        push    dx
        mov     ah, 09H
        lea     dx, MENS_MENU
        int     21H                             ;mensagem
        pop     dx
        pop     ax
        ret
ESCREVE_MENU  ENDP

;*****************************************************************************************************************
;Subrotina:      ESCREVE_FINAL
;               Escreve a mensagem de despedida
;*****************************************************************************************************************
PUBLIC ESCREVE_FINAL
ESCREVE_FINAL  PROC
        push    ax
        push    dx
        mov     ah, 09H
        lea     dx, MENS_FINAL
        int     21H                             ;mensagem
        pop     dx
        pop     ax
        ret
ESCREVE_FINAL  ENDP

;*****************************************************************************************************************
;Subrotina:      ESCREVE_M0_H
;               Escreve a mensagem correspondente ao motor em movimento
;*****************************************************************************************************************
PUBLIC ESCREVE_M0_H
ESCREVE_M0_H  PROC
        push    ax
        push    dx
        push    bx                              ;necessário pelo caractere colorido
        push    cx                              ;necessário pelo caractere colorido
        mov     ah, 09H
        lea     dx, MENS_M0_H
```

```
        int     21H                             ;mensagem
        push    bp                              ;cria caractere colorido
        mov     ah, 09H                         ;no final da mensagem
        mov     al, 16H
        mov     bl, 44H                         ;define atributo, cor = vermelho
        mov     cx, 0009H
        int     10H
        pop     bp
        pop     cx
        pop     bx
        pop     dx
        pop     ax
        ret
ESCREVE_M0_H    ENDP

;*****************************************************************************************************************
;Subrotina:      ESCREVE_M0_AH
;               Escreve a mensagem correspondente ao motor em movimento
;*****************************************************************************************************************
PUBLIC ESCREVE_M0_AH
ESCREVE_M0_AH PROC
        push    ax
        push    dx
        push    bx                              ;necessário pelo caractere colorido
        push    cx                              ;necessário pelo caractere colorido
        mov     ah, 09H
        lea     dx, MENS_M0_AH
        int     21H                             ;mensagem
        push    bp                              ;cria caractere colorido
        mov     ah, 09H                         ;no final da mensagem
        mov     al, 16H
        mov     bl, 44H                         ;define atributo, cor = vermelho
        mov     cx, 0009H
        int     10H
        pop     bp
        pop     cx
        pop     bx
        pop     dx
        pop     ax
        ret
ESCREVE_M0_AH ENDP

;*****************************************************************************************************************
;Subrotina:      ESCREVE_M1_H
;               Escreve a mensagem correspondente ao motor em movimento
;*****************************************************************************************************************
PUBLIC ESCREVE_M1_H
ESCREVE_M1_H    PROC
        push    ax
        push    dx
        push    bx                              ;necessário pelo caractere colorido
        push    cx                              ;necessário pelo caractere colorido
        mov     ah, 09H
        lea     dx, MENS_M1_H
        int     21H                             ;mensagem
        push    bp                              ;cria caractere colorido
        mov     ah, 09H                         ;no final da mensagem
        mov     al, 16H
        mov     bl, 11H                         ;define atributo, cor = azul escuro
        mov     cx, 0009H
        int     10H
        pop     bp
        pop     cx
        pop     bx
        pop     dx
        pop     ax
        ret
ESCREVE_M1_H    ENDP

;*****************************************************************************************************************
```

```
;Subrotina:          ESCREVE_M1_AH
;                    Escreve a mensagem correspondente ao motor em movimento
;*********************************************************************************************************************
PUBLIC ESCREVE_M1_AH
ESCREVE_M1_AH PROC
        push    ax
        push    dx
        push    bx                              ;necessário pelo caractere colorido
        push    cx                              ;necessário pelo caractere colorido
        mov     ah, 09H
        lea     dx, MENS_M1_AH
        int     21H                             ;mensagem
        push    bp                              ;cria caractere colorido
        mov     ah, 09H                         ;no final da mensagem
        mov     al, 16H
        mov     bl, 11H                         ;define atributo, cor = azul escuro
        mov     cx, 0009H
        int     10H
        pop     bp
        pop     cx
        pop     bx
        pop     dx
        pop     ax
        ret
ESCREVE_M1_AH ENDP

;*********************************************************************************************************************
;Subrotina:          ESCREVE_M2_H
;                    Escreve a mensagem correspondente ao motor em movimento
;*********************************************************************************************************************
PUBLIC ESCREVE_M2_H
ESCREVE_M2_H  PROC
        push    ax
        push    dx
        push    bx                              ;necessário pelo caractere colorido
        push    cx                              ;necessário pelo caractere colorido
        mov     ah, 09H
        lea     dx, MENS_M2_H
        int     21H                             ;mensagem
        push    bp                              ;cria caractere colorido
        mov     ah, 09H                         ;no final da mensagem
        mov     al, 16H
        mov     bl, 22H                         ;define atributo, cor = verde claro
        mov     cx, 0009H
        int     10H
        pop     bp
        pop     cx
        pop     bx
        pop     dx
        pop     ax
        ret
ESCREVE_M2_H  ENDP

;*********************************************************************************************************************
;Subrotina:          ESCREVE_M2_AH
;                    Escreve a mensagem correspondente ao motor em movimento
;************************************************* *****************************************************************
PUBLIC ESCREVE_M2_AH
ESCREVE_M2_AH PROC
        push    ax
        push    dx
        push    bx                              ;necessário pelo caractere colorido
        push    cx                              ;necessário pelo caractere colorido
        mov     ah, 09H
        lea     dx, MENS_M2_AH
        int     21H                             ;mensagem
        push    bp                              ;cria caractere colorido
        mov     ah, 09H                         ;no final da mensagem
        mov     al, 16H
        mov     bl, 22H                         ;define atributo, cor = verde claro
```

```
        mov     cx, 0009H
        int     10H
        pop     bp
        pop     cx
        pop     bx
        pop     dx
        pop     ax
        ret
ESCREVE_M2_AH ENDP

;*****************************************************************************************************************
;Subrotina:      ESCREVE_M3_H
;               Escreve a mensagem correspondente ao motor em movimento
;*****************************************************************************************************************
PUBLIC ESCREVE_M3_H
ESCREVE_M3_H    PROC
        push    ax
        push    dx
        push    bx                              ;necessário pelo caractere colorido
        push    cx                              ;necessário pelo caractere colorido
        mov     ah, 09H
        lea     dx, MENS_M3_H
        int     21H                             ;mensagem
        push    bp                              ;cria caractere colorido
        mov     ah, 09H                         ;no final da mensagem
        mov     al, 16H
        mov     bl, 55H                         ;define atributo, cor = rosa
        mov     cx, 0009H
        int     10H
        pop     bp
        pop     cx
        pop     bx
        pop     dx
        pop     ax
        ret
ESCREVE_M3_H    ENDP

;*****************************************************************************************************************
;Subrotina:      ESCREVE_M3_AH
;               Escreve a mensagem correspondente ao motor em movimento
;*****************************************************************************************************************
PUBLIC ESCREVE_M3_AH
ESCREVE_M3_AH   PROC
        push    ax
        push    dx
        push    bx                              ;necessário pelo caractere colorido
        push    cx                              ;necessário pelo caractere colorido
        mov     ah, 09H
        lea     dx, MENS_M3_AH
        int     21H                             ;mensagem
        push    bp                              ;cria caractere colorido
        mov     ah, 09H                         ;no final da mensagem
        mov     al, 16H
        mov     bl, 55H                         ;define atributo, cor = rosa
        mov     cx, 0009H
        int     10H
        pop     bp
        pop     cx
        pop     bx
        pop     dx
        pop     ax
        ret
ESCREVE_M3_AH   ENDP

;*****************************************************************************************************************
;Subrotina:      ESCREVE_M0_STOP
;               Escreve a mensagem correspondente ao motor parado
;*****************************************************************************************************************
PUBLIC ESCREVE_M0_STOP
ESCREVE_M0_STOP         PROC
```

```
        push    ax
        push    dx
        push    bx                              ;necessário pelo caractere colorido
        push    cx                              ;necessário pelo caractere colorido
        mov     ah, 09H
        lea     dx, MENS_M0_STOPED
        int     21H                             ;mensagem
        push    bp                              ;cria caractere colorido
        mov     ah, 09H                         ;no final da mensagem
        mov     al, 16H
        mov     bl, 44H                         ;define atributo, cor = vermelho
        mov     cx, 0002H
        int     10H
        pop     bp
        pop     cx
        pop     bx
        pop     dx
        pop     ax
        ret
ESCREVE_M0_STOP         ENDP

;*****************************************************************************************************************
;Subrotina:             ESCREVE_M1_STOP
;                       Escreve a mensagem correspondente ao motor parado
;*****************************************************************************************************************
PUBLIC ESCREVE_M1_STOP
ESCREVE_M1_STOP         PROC
        push    ax
        push    dx
        push    bx                              ;necessário pelo caractere colorido
        push    cx                              ;necessário pelo caractere colorido
        mov     ah, 09H
        lea     dx, MENS_M1_STOPED
        int     21H                             ;mensagem
        push    bp                              ;cria caractere colorido
        mov     ah, 09H                         ;no final da mensagem
        mov     al, 16H
        mov     bl, 11H                         ;define atributo, cor = azul escuro
        mov     cx, 0002H
        int     10H
        pop     bp
        pop     cx
        pop     bx
        pop     dx
        pop     ax
        ret
ESCREVE_M1_STOP         ENDP

;*****************************************************************************************************************
;Subrotina:             ESCREVE_M2_STOP
;                       Escreve a mensagem correspondente ao motor parado
;*****************************************************************************************************************
PUBLIC ESCREVE_M2_STOP
ESCREVE_M2_STOP         PROC
        push    ax
        push    dx
        push    bx                              ;necessário pelo caractere colorido
        push    cx                              ;necessário pelo caractere colorido
        mov     ah, 09H
        lea     dx, MENS_M2_STOPED
        int     21H                             ;mensagem
        push    bp                              ;cria caractere colorido
        mov     ah, 09H                         ;no final da mensagem
        mov     al, 16H
        mov     bl, 22H                         ;define atributo, cor = verde claro
        mov     cx, 0002H
        int     10H
        pop     bp
        pop     cx
        pop     bx
```

```
        pop     dx
        pop     ax
        ret
ESCREVE_M2_STOP         ENDP

;*****************************************************************************************************************
;Subrotina:             ESCREVE_M3_STOP
;                       Escreve a mensagem correspondente ao motor parado
;*****************************************************************************************************************
PUBLIC ESCREVE_M3_STOP
ESCREVE_M3_STOP         PROC
        push    ax
        push    dx
        push    bx                              ;necessário pelo caractere colorido
        push    cx                              ;necessário pelo caractere colorido
        mov     ah, 09H
        lea     dx, MENS_M3_STOPED
        int     21H                             ;mensagem
        push    bp                              ;cria caractere colorido
        mov     ah, 09H                         ;no final da mensagem
        mov     al, 16H
        mov     bl, 55H                         ;define atributo, cor = rosa
        mov     cx, 0002H
        int     10H
        pop     bp
        pop     cx
        pop     bx
        pop     dx
        pop     ax
        ret
ESCREVE_M3_STOP         ENDP

;*****************************************************************************************************************
;Subrotina:             ESCREVE_G_ABERTA
;                       Escreve a mensagem correspondente ao status da garra
;*****************************************************************************************************************
PUBLIC ESCREVE_G_ABERTA
ESCREVE_G_ABERTA PROC
        push    ax
        push    dx
        push    bx                              ;necessário pelo caractere colorido
        push    cx                              ;necessário pelo caractere colorido
        mov     ah, 09H
        lea     dx, MENS_G_OPENED
        int     21H                             ;mensagem
        push    bp                              ;cria caractere colorido
        mov     ah, 09H                         ;no final da mensagem
        mov     al, 16H
        mov     bl, 66H                         ;define atributo, cor = laranja
        mov     cx, 0002H
        int     10H
        pop     bp
        pop     cx
        pop     bx
        pop     dx
        pop     ax
        ret
ESCREVE_G_ABERTA ENDP

;*****************************************************************************************************************
;Subrotina:             ESCREVE_G_FECHADA
;                       Escreve a mensagem correspondente ao status da garra
;*****************************************************************************************************************
PUBLIC ESCREVE_G_FECHADA
ESCREVE_G_FECHADA PROC
        push    ax
        push    dx
        push    bx                              ;necessário pelo caractere colorido
        push    cx                              ;necessário pelo caractere colorido
        mov     ah, 09H
```

```
        lea     dx, MENS_G_CLOSED
        int     21H                     ;mensagem
        push    bp                      ;cria caractere colorido
        mov     ah, 09H                 ;no final da mensagem
        mov     al, 16H
        mov     bl, 66H                 ;define atributo, cor = laranja
        mov     cx, 0009H
        int     10H
        pop     bp
        pop     cx
        pop     bx
        pop     dx
        pop     ax
        ret
ESCREVE_G_FECHADA ENDP

        END     ANA1                    ;finaliza o programa
```

5.3 Conjunto de INSTRUÇÕES BÁSICAS dos microprocessadores das sub-famílias Intel 80x86, 80x88 e Pentium

Observações:

Para compactar as tabelas apresentadas sem perda de informação é adotada a seguinte notação para as instruções:

- Um valor numérico de 8 bits qualquer é chamado "byte". Este possui o seu bit +significativo (colocado a esquerda) chamado D7 e o seu bit - significativo (colocado a direita) D0.

- Um valor numérico de 4 bits é chamado "nibble", existindo assim em um byte, o nibble +significativo (colocado a esquerda) e o nibble – significativo (colocado a direita).

- Um valor numérico de 16 bits qualquer é chamado de "Word" ou "dois bytes", existindo assim o byte +significativo e o byte –significativo, podendo também representar endereços.

- Os registradores de 8 bit s especificados genericamente nas instruções são: AH, AL, BH, BL, CH, CL, DH, DL, [BX] e [endereço].

- Os registradores de 16 bits especificados genericamente nas instruções são: AX, BX, CX e DX.

- Ao utilizar combinações de uma determinada instrução lembre-se que nunca pode existir [BX] e [endereço] simultaneamente no mesmo mnemônico.

INSTRUÇÕES PARA ENTRADA DE DADOS EM REGISTRADORES DE 8 BITS

Mnemônico:	Descrição da instrução:
MOV AL, byte	AL ← byte qualquer ; carrega um byte em AL
MOV BL, byte	BL ← byte qualquer ; carrega um byte em BL
	Demais registradores de 8 bits (...H ou ...L), [BX] ou [endereço] são válidos

INSTRUÇÕES PARA ENTRADA DE DADOS EM REGISTRADORES DE 16 BITS

Mnemônico:	Descrição da instrução:
MOV AX, dois bytes	AX ← Word ou AH← (byte +significativo) e AL ← byte (–significativo)
MOV BX, dois bytes	BX ← Word ou BH← (byte +significativo) e BL ← byte (–significativo)
	Demais registradores de 16 bits (...X) são válidos

INSTRUÇÕES DE TRANSFERÊNCIA DE DADOS ENTRE REGISTRADORES DE 8 BITS

Mnemônico:	Descrição da instrução:
MOV AL, CL	AL ← CL ; carrega o conteúdo de CL em AL
MOV CL, BL	CL ← BL ; carrega o conteúdo de BL em CL
	Demais combinações entre registradores de 8 bits (...H e ...L) , [BX] ou [endereço] são válidas

INSTRUÇÕES DE TRANSFERÊNCIA DE DADOS ENTRE REGISTRADORES DE 16 BITS

Mnemônico:	Descrição da instrução:
MOV AX, AX	AX ← AX ; carrega o conteúdo de AX em AX
MOV DX, BX	DX ← BX ; carrega o conteúdo de BX em DX
	Demais combinações entre registradores de 16 bits (...X) são válidas

INSTRUÇÕES DE PERMUTA DE DADOS ENTRE REGISTRADORES DE 8 BITS

Mnemônico:	Descrição da instrução:
XCHG CL, BH	CL ← → BH ; troca os conteúdos dos registradores AL e AL
XCHG DL, AL	DL ← → AL ; troca os conteúdos dos registradores DL e AL
	Demais combinações entre registradores de 8 bits (...H e ...L) , [BX] ou [endereço] são válidas

INSTRUÇÕES DE PERMUTA DE DADOS ENTRE REGISTRADORES DE 16 BITS

Mnemônico:	Descrição da instrução:
XCHG AX, DX	AX ← →DX ; troca os conteúdos dos registradores AX e DX
XCHG BX, CX	BX ← →CX ; troca os conteúdos dos registradores BX e CX
	Demais combinações entre registradores de 16 bits (...X) são válidas

INSTRUÇÕES ARITMÉTICAS ENTRE REGISTRADORES DE 8 BITS

Mnemônico:	Descrição da instrução:
ADD AL, BL	AL ← AL + BL
ADD DH, AL	DH ← DH + AL
	Demais combinações entre registradores de 8 bits (...H e ...L) , [BX] ou [endereço] são válidas
ADC AL, AH	AL ← AL + AH + carry flag
ADC DL, CL	DL ← DL + CL + carry flag
	Demais combinações entre registradores de 8 bits (...H e ...L) , [BX] ou [endereço] são válidas
SUB AL, BL	AL ← AL + BL
SUB CL, DL	CL ← CL + DL
	Demais combinações entre registradores de 8 bits (...H e ...L) , [BX] ou [endereço] são válidas
SBB AL, AH	AL ← AL - AH - carry flag
SBB DL, BL	DL ← DL - BL - carry flag
	Demais combinações entre registradores de 8 bits (...H e ...L) , [BX] ou [endereço] são válidas
MUL BL	AX ← AL . BL
MUL CL	AX ← AL . CL
	Demais registradores de 8 bits (...H e ...L) são válidos para o multiplicador Os registradores AL (multiplicando) e AX (produto) sempre são usados
DIV DH	AL ← AX / DH ; AH ← resto da divisão
DIV CL	AL ← AX / CL ; AH ← resto da divisão
	Demais registradores de 8 bits (...H e ...L) , são válidos para o divisor Os registradores AX (dividendo) e AL (quociente) e AH (resto) sempre são usados Instrução indefinida para divisor igual a zero

INSTRUÇÕES ARITMÉTICAS ENTRE REGISTRADORES E DADOS DE 8 BITS

Mnemônico:	Descrição da instrução:
ADD AL, byte	AL ← AL + byte
ADC DH, byte	DH ← DH + byte + carry flag
	Demais combinações entre registradores de 8 bits (...H e ...L) , [BX] ou [endereço] são válidas
SUB CL, byte	CL ← CL – byte
SBB BL, byte	BL ← BL – byte - carry flag
	Demais combinações entre registradores de 8 bits (...H e ...L) , [BX] ou [endereço] são válidas

INSTRUÇÕES ARITMÉTICAS SOBRE REGISTRADORES DE 8 BITS

Mnemônico:	Descrição da instrução:
INC AL	AL ← AL + 1
	Demais combinações entre registradores de 8 bits (...H e ...L) , [BX] ou [endereço] são válidas
DEC AL	AL ← AL – 1
	Demais combinações entre registradores de 8 bits (...H e ...L) , [BX] ou [endereço] são válidas
NEG CL	CL ← complemento de dois do valor anterior Torna o conteúdo de CL negativo
	Demais combinações entre registradores de 8 bits (...H e ...L) , [BX] ou [endereço] são válidas

INSTRUÇÕES ARITMÉTICAS ENTRE REGISTRADORES DE 16 BITS

Mnemônico:	Descrição da instrução:
ADD AX, BX	AX ← AX + BX
ADD DX, CX	DX ← DX + CX
	Demais combinações entre registradores de 16 bits (...X) são válidas
ADC AX, BX	AX ← AX + BX + carry flag
ADC BX, AX	BX ← BX + AX + carry flag
	Demais combinações entre registradores de 16 bits (...X) são válidas
SUB AX, CX	AX ← AX - CX
SUB BX, AX	BX ← BX - AX
	Demais combinações entre registradores de 16 bits (...X) são válidas
SBB AX, DX	AX ← AX - DX – carry flag
SBB BX, AX	BX ← BX - AX – carry flag
	Demais combinações entre registradores de 16 bits (...X) são válidas
MUL BX MUL CX	DX : AX ← AX . BX DX : AX ← AX . CX
	Demais registradores de 16 bits (...X) são válidos para o multiplicador Os registradores AX (multiplicando) e DX:AX (produto) sempre são usados
DIV BX	AX ← DX:AX / BX; DX ← resto da divisão
DIV CX	AX ← DX:AX / CX ; DX ← resto da divisão
	Demais registradores de 16 bits (...X) são válidos para o divisor Os registradores DX:AX (dividendo), AX (quociente) e DX (resto) sempre são usados. Instrução indefinida para divisor igual a zero

INSTRUÇÕES ARITMÉTICAS ENTRE DADOS E REGISTRADORES DE 16 BITS

Mnemônico:	Descrição da instrução:
ADD AX, word	AX ← AX + word
ADD BX, word	BX ← BX + word
	Demais registradores de 16 bits (...X) são válidos
ADC BX, word	BX ← BX + word + carry flag
ADC CX, word	CX ← CX + word + carry flag
	Demais registradores de 16 bits (...X) são válidos
SUB CX, word	CX ← CX - word
SUB DX, word	DX ← DX - word
	Demais registradores de 16 bits (...X) são válidos
SBB BX, word	BX ← BX – word – carry flag
SBB CX, word	CX ← CX – word – carry flag
	Demais registradores de 16 bits (...X) são válidos

INSTRUÇÕES ARITMÉTICAS SOBRE REGISTRADORES DE 16 BITS

Mnemônico:	Descrição da instrução:
INC AX	AX ← AX + 1
	Demais registradores de 16 bits (...X) são válidos
DEC BX	BX ← BX - 1
	Demais registradores de 16 bits (...X) são válidos
NEG DX	DX ← complemento de dois de DX, torna o conteúdo de DX negativo
	Demais registradores de 16 bits (...X) são válidos

INSTRUÇÕES LÓGICAS ENTRE REGISTRADORES DE 8 BITS

Mnemônico:	Descrição da instrução:
AND AL, BL	AL ← AL . BL
AND DH, CL	DH ← DH . CL
	Demais combinações entre registradores de 8 bits (...H ou ...L) , [BX] ou [endereço] são válidas
OR BL, BH	BL ← BL + BH
OR DL, AL	DL ← DL + AL
	Demais combinações entre registradores de 8 bits (...H ou ...L) , [BX] ou [endereço] são válidas
XOR AH, BH	AH ← AH (exclusive or) BH
XOR AH, CL	AH ← AH (exclusive or) CL
	Demais combinações entre registradores de 8 bits (...H ou ...L) , [BX] ou [endereço] são válidas
XOR DL, BH	DL ← DL (exclusive or) BH
XOR DL, CL	DL ← DL (exclusive or) CL
	Demais combinações entre registradores de 8 bits (...H ou ...L) , [BX] ou [endereço] são válidas
NOT AH	AH ← (AH)' ; complementa os bits do registrador
NOT BL	BL ← (BL)' ; complementa os bits do registrador
	Demais registradores de 8 bits (...H ou ...L) , [BX] ou [endereço] são válidos

INSTRUÇÕES LÓGICAS ENTRE REGISTRADORES E DADOS DE 8 BITS

Mnemônico:	Descrição da instrução:
AND AL, byte	AL ← AL . byte
	Demais registradores de 8 bits (...H ou ...L) , [BX] ou [endereço] são válidos
OR BL, byte	BL ← BL + byte
	Demais registradores de 8 bits (...H ou ...L) , [BX] ou [endereço] são válidos
XOR CL, byte	CL ← CL (exclusive or) byte
	Demais registradores de 8 bits (...H ou ...L) , [BX] ou [endereço] são válidos

INSTRUÇÕES LÓGICAS ENTRE REGISTRADORES DE 16 BITS

Mnemônico:	Descrição da instrução:
AND AX, BX	AX ← AX . BX
	Demais combinações entre registradores de 16 bits (...X) são válidas
AND BX, CX	BX ← BX . CX
	Demais combinações entre registradores de 16 bits (...X) são válidas
OR BX, CX	BX ← BX + CX
	Demais combinações entre registradores de 16 bits (...X) são válidas
XOR CX, AX	CX ← CX (exclusive or) AX
	Demais combinações entre registradores de 16 bits (...X) são válidas
NOT DX	DX ← (DX)' ; complementa os bits do registrador
	Demais registradores de 16 bits (...X) são válidos

INSTRUÇÕES LÓGICAS ENTRE REGISTRADORES E DADOS DE 16 BITS

Mnemônico:	Descrição da instrução:
AND AX, word	AX ← AX . word
	Demais registradores de 16 bits (...X) são válidos
OR BX, word	BX ← BX + word
	Demais registradores de 16 bits (...X) são válidos
XOR DX, word	DX ← DX (exclusive or) word
	Demais registradores de 16 bits (...X) são válidos

INSTRUÇÕES DE TRANSFERÊNCIA DE DADOS ENTRE REGISTRADORES E MEMÓRIA

Mnemônico:	Descrição da instrução:
MOV AL, [endereço]	AL ← [endereço] ; carrega o conteúdo do endereço em AL
MOV BH, [endereço]	BH ← [endereço] ; carrega o conteúdo do endereço em BH
	Demais registradores de 8 bits (...H ou ...L) são válidos
MOV [endereço], AH	[endereço] ← AH ; carrega o conteúdo de AH no endereço
MOV [endereço], BL	[endereço] ← BL ; carrega o conteúdo de BL no endereço
	Demais registradores de 8 bits (...H ou ...L) são válidos
MOV AX, [endereço]	AL ← [endereço] e AH ← [endereço + 1]
MOV BX, [endereço]	BL ← [endereço] e BH ← [endereço + 1]
	Demais registradores de 16 bits (...X) são válidos
MOV [endereço], BX	[endereço] ← BL e [endereço + 1] ← BH
MOV [endereço], CX	[endereço] ← CL e [endereço + 1] ← CH
	Demais registradores de 16 bits (...X) são válidos
MOV AL, [BX]	AL ← [BX] ; carrega em AL o conteúdo endereçado por BX
MOV CH, [BX]	CH ← [BX] ; carrega em CH o conteúdo endereçado por BX
	Demais registradores de 8 bits (...H ou ...L) são válidos Somente [BX] é válido
MOV [BX], AL	[BX] ← AL ; carrega AL no endereço indicado por BX
MOV [BX], CH	[BX] ← CH ; carrega CH no endereço indicado por BX
	Demais registradores de 8 bits (...H ou ...L) são válidos Somente [BX] é válido

INSTRUÇÕES DE ROTAÇÃO DE UM BIT, EM REGISTRADORES DE 8 BITS

Mnemônico:	Descrição da instrução:
ROL AL, 1	Rotação de AL, para a esquerda de um bit, copiando b_7 na carry flag $b_7 \leftarrow b_6 \leftarrow b_5 \leftarrow b_4 \leftarrow b_3 \leftarrow b_2 \leftarrow b_1 \leftarrow b_0 \leftarrow b_7$ e carry flag ← b_7
ROL BL, 1	Rotação de BL, para a esquerda de um bit, copiando b_7 na carry flag $b_7 \leftarrow b_6 \leftarrow b_5 \leftarrow b_4 \leftarrow b_3 \leftarrow b_2 \leftarrow b_1 \leftarrow b_0 \leftarrow b_7$ e carry flag ← b_7
	Demais registradores de 8 bits (...H ou ...L) , [BX] ou [endereço] são válidos
ROR AH, 1	Rotação de AH, para a direita de um bit, copiando b_0 na carry flag $b_7 \rightarrow b_6 \rightarrow b_5 \rightarrow b_4 \rightarrow b_3 \rightarrow b_2 \rightarrow b_1 \rightarrow b_0 \rightarrow b_7$ e carry flag ← b_0
ROR CL, 1	Rotação de CL, para a direita de um bit, copiando b_0 na carry flag $b_7 \rightarrow b_6 \rightarrow b_5 \rightarrow b_4 \rightarrow b_3 \rightarrow b_2 \rightarrow b_1 \rightarrow b_0 \rightarrow b_7$ e carry flag ← b_0
	Demais registradores de 8 bits (...H ou ...L) , [BX] ou [endereço] são válidos
RCL AL, 1	Rotação de AL, para a esquerda de um bit, através da carry flag $b_7 \leftarrow b_6 \leftarrow b_5 \leftarrow b_4 \leftarrow b_3 \leftarrow b_2 \leftarrow b_1 \leftarrow b_0 \leftarrow$ carry flag ← b_7
RCL BH, 1	Rotação de BH, para a esquerda de um bit, através da carry flag $b_7 \leftarrow b_6 \leftarrow b_5 \leftarrow b_4 \leftarrow b_3 \leftarrow b_2 \leftarrow b_1 \leftarrow b_0 \leftarrow$ carry flag ← b_7 Demais registradores de 8 bits (...H ou ...L) , [BX] ou [endereço] são válidos
RCR AH, 1	Rotação de AH, para a direita de um bit, através da carry flag $b_7 \rightarrow b_6 \rightarrow b_5 \rightarrow b_4 \rightarrow b_3 \rightarrow b_2 \rightarrow b_1 \rightarrow b_0 \rightarrow$ carry flag → b_7
RCR BL, 1	Rotação de BL, para a direita de um bit, através da carry flag $b_7 \rightarrow b_6 \rightarrow b_5 \rightarrow b_4 \rightarrow b_3 \rightarrow b_2 \rightarrow b_1 \rightarrow b_0 \rightarrow$ carry flag → b_7
	Demais registradores de 8 bits (...H ou ...L) , [BX] ou [endereço] são válidos

INSTRUÇÕES DE ROTAÇÃO DE "N" BITS, EM REGISTRADORES DE 8 BITS

Mnemônico:	Descrição da instrução:
ROL AL, CL	Rotação de AL, para a esquerda de um número de bits indicado em CL e copiando b_7 na carry flag
ROL BH, CL	Rotação de BH, para a esquerda de um número de bits indicado em CL e copiando b_7 na carry flag
	Demais registradores de 8 bits (...H ou ...L) são válidos, [BX] ou [endereço] O número de bits rotacionados sempre deve ficar em CL
ROR AH, CL	Rotação de AH, para a direita de um número de bits indicado em CL e copiando b_0 na carry flag
ROR BL, CL	Rotação de BL, para a direita de um número de bits indicado em CL e copiando b_0 na carry flag
	Demais registradores de 8 bits (...H ou ...L) são válidos, [BX] ou [endereço] O número de bits rotacionados sempre deve ficar em CL
RCL BL, CL	Rotação de BL, para a esquerda de um número de bits indicado em CL Através da carry flag
RCL BH, CL	Rotação de BH, para a esquerda de um número de bits indicado em CL Através da carry flag
	Demais registradores de 8 bits (...H ou ...L) são válidos, [BX] ou [endereço] O número de bits rotacionados sempre deve ficar em CL
RCR AH, CL	Rotação de AH, para a direita de um número de bits indicado em CL Através da carry flag
RCR BL, CL	Rotação de BL, para a direita de um número de bits indicado em CL Através da carry flag
	Demais registradores de 8 bits (...H ou ...L) , [BX] ou [endereço] são válidos O número de bits rotacionados sempre deve ficar em CL

INSTRUÇÕES DE ROTAÇÃO DE UM BIT, EM REGISTRADORES DE 16 BITS

Mnemônico:	Descrição da instrução:
ROL AX, 1	Rotação de AX, para a esquerda de um bit, copiando b_{15} na carry flag; $b_{15}\leftarrow b_{14}\leftarrow b_{13}\leftarrow b_{12}\leftarrow b_{11}\leftarrow b_{10}\leftarrow b_9\leftarrow b_8\leftarrow b_7\leftarrow b_6\leftarrow b_5\leftarrow b_4\leftarrow b_3\leftarrow b_2\leftarrow b_1\leftarrow b_0\leftarrow b_{15}$ e e carry flag $\leftarrow b_{15}$
ROL BX, 1	Rotação de BX, para a esquerda de um bit, copiando b_{15} na carry flag; $b_{15}\leftarrow b_{14}\leftarrow b_{13}\leftarrow b_{12}\leftarrow b_{11}\leftarrow b_{10}\leftarrow b_9\leftarrow b_8\leftarrow b_7\leftarrow b_6\leftarrow b_5\leftarrow b_4\leftarrow b_3\leftarrow b_2\leftarrow b_1\leftarrow b_0\leftarrow b_{15}$ e e carry flag $\leftarrow b_{15}$
	Demais registradores de 16 bits (...X) são válidos
ROR CX, 1	Rotação de CX, para a direita de um bit, copiando b_0 na carry flag; $B_{15}\rightarrow b_{14}\rightarrow b_{13}\rightarrow b_{12}\rightarrow b_{11}\rightarrow b_{10}\rightarrow b_9\rightarrow b_8\rightarrow b_7\rightarrow b_6\rightarrow b_5\rightarrow b_4\rightarrow b_3\rightarrow b_2\rightarrow b_1\rightarrow b_0\rightarrow b_{15}$ e e carry flag $\leftarrow b_0$
ROR DX, 1	Rotação de DX, para a direita de um bit, copiando b_0 na carry flag; $B_{15}\rightarrow b_{14}\rightarrow b_{13}\rightarrow b_{12}\rightarrow b_{11}\rightarrow b_{10}\rightarrow b_9\rightarrow b_8\rightarrow b_7\rightarrow b_6\rightarrow b_5\rightarrow b_4\rightarrow b_3\rightarrow b_2\rightarrow b_1\rightarrow b_0\rightarrow b_{15}$ e e carry flag $\leftarrow b_0$
	Demais registradores de 16 bits (...X) são válidos
RCL AX, 1	Rotação de AX, para a esquerda de um bit, através da carry flag; $b_{15}\leftarrow b_{14}\leftarrow b_{13}\leftarrow b_{12}\leftarrow b_{11}\leftarrow b_{10}\leftarrow b_9\leftarrow b_8\leftarrow b_7\leftarrow b_6\leftarrow b_5\leftarrow b_4\leftarrow b_3\leftarrow b_2\leftarrow b_1\leftarrow b_0$ e $b_0\leftarrow$carry flag $\leftarrow b_{15}$
RCL BX, 1	Rotação de BX, para a esquerda de um bit, através da carry flag; $b_{15}\leftarrow b_{14}\leftarrow b_{13}\leftarrow b_{12}\leftarrow b_{11}\leftarrow b_{10}\leftarrow b_9\leftarrow b_8\leftarrow b_7\leftarrow b_6\leftarrow b_5\leftarrow b_4\leftarrow b_3\leftarrow b_2\leftarrow b_1\leftarrow b_0$ e $b_0\leftarrow$carry flag $\leftarrow b_{15}$
	Demais registradores de 16 bits (...X) são válidos
RCR BX, 1	Rotação de BX, para a direita de um bit, através da carry flag; $b_{15}\rightarrow b_{14}\rightarrow b_{13}\rightarrow b_{12}\rightarrow b_{11}\rightarrow b_{10}\rightarrow b_9\rightarrow b_8\rightarrow b_7\rightarrow b_6\rightarrow b_5\rightarrow b_4\rightarrow b_3\rightarrow b_2\rightarrow b_1\rightarrow b_0$ e $b_0\rightarrow$ carry flag $\rightarrow b_{15}$
RCR CX, 1	Rotação de CX, para a direita de um bit, através da carry flag; $b_{15}\rightarrow b_{14}\rightarrow b_{13}\rightarrow b_{12}\rightarrow b_{11}\rightarrow b_{10}\rightarrow b_9\rightarrow b_8\rightarrow b_7\rightarrow b_6\rightarrow b_5\rightarrow b_4\rightarrow b_3\rightarrow b_2\rightarrow b_1\rightarrow b_0$ e $b_0\rightarrow$ carry flag $\rightarrow b_{15}$
	Demais registradores de 16 bits (...X) são válidos

INSTRUÇÕES DE ROTAÇÃO DE "N" BITS, EM REGISTRADORES DE 16 BITS

Mnemônico:	Descrição da instrução:
ROL AX, CL	Rotação de AX, para a esquerda de um número de bits indicado em CL e copiando b_{15} na carry flag
ROL BX, CL	Rotação de BX, para a esquerda de um número de bits indicado em CL e copiando b_{15} na carry flag
	Demais registradores de 16 bits (...X) são válidos O número de bits rotacionados sempre deve ficar em CL
ROR AX, CL	Rotação de AX, para a direita de um número de bits indicado em CL e copiando b_0 na carry flag
ROR BX, CL	Rotação de BX, para a direita de um número de bits indicado em CL e copiando b_0 na carry flag
	Demais registradores de 16 bits (...X) são válidos O número de bits rotacionados sempre deve ficar em CL
RCL BX, CL	Rotação de BX, para a esquerda de um número de bits indicado em CL Através da carry flag
RCL CX, CL	Rotação de CX, para a esquerda de um número de bits indicado em CL Através da carry flag
	Demais registradores de 16 bits (...X) são válidos O número de bits rotacionados sempre deve ficar em CL
RCR AX, CL	Rotação de AX, para a direita de um número de bits indicado em CL Através da carry flag
RCR BX, CL	Rotação de BX, para a direita de um número de bits indicado em CL Através da carry flag
	Demais registradores de 16 bits (...X) são válidos O número de bits rotacionados sempre deve ficar em CL

INSTRUÇÕES DE SAÍDA DE DADOS

Mnemônico	Descrição da instrução:
OUT porta, AL	(porta) ← AL ; saída de dados, envia o conteúdo de AL para a porta cujo endereço é fornecido (endereço de 8 bits)
OUT DX, AL	(DX) ← AL ; envia o conteúdo de AL para a porta cujo endereço é especificado em DX (endereço de 16 bits)
OUT porta, AX	(porta) ← AX ; saída de dados, envia o conteúdo de AX para a porta cujo endereço é fornecido (endereço de 8 bits)
OUT DX, AX	(DX) ← AX ; envia o conteúdo de AX para a porta cujo endereço é especificado em DX (endereço de 16 bits)

INSTRUÇÕES DE ENTRADA DE DADOS

Mnemônico	Descrição da instrução:
IN AL, porta	AL ← (porta) ; entrada de dados; Lê o byte presente na porta cujo endereço é fornecido, e o carrega em AL (endereço de 8 bits)
IN AL, DX	AL ← (DX) ; Lê o byte presente na porta cujo endereço é especificado em DX e o carrega em AL (endereço de 16 bits)
IN AX, porta	AX ← (porta) ; entrada de dados; Lê o byte presente na porta cujo endereço é fornecido, e o carrega em AX (endereço de 8 bits)
IN AX, DX	AX ← (DX) ; Lê o byte presente na porta cujo endereço é especificado em DX e o carrega em AX (endereço de 16 bits)

INSTRUÇÕES PARA COMPARAÇÕES ENTRE REGISTRADORES DE 8 BITS

Mnemônico:	Descrição da instrução:
CMP AL, AH	(flags) ← (status da comparação de AL com AH)
CMP BL, CL	(flags) ← (status da comparação de BL com CL)
	Demais combinações entre registradores de 8 bits (...H e ...L) , [BX] ou [endereço] são válidas

INSTRUÇÕES PARA COMPARAÇÕES ENTRE REGISTRADORES E DADOS DE 8 BITS

Mnemônico:	Descrição da instrução:
CMP AL, byte	(flags) ← (status da comparação de AL com byte)
CMP BL, byte	(flags) ← (status da comparação de BL com byte)
	Demais registradores de 8 bits (...H ou ...L) , [BX] ou [endereço] são válidos

INSTRUÇÕES PARA COMPARAÇÕES ENTRE REGISTRADORES DE 16 BITS

Mnemônico:	Descrição da instrução:
CMP AX, BX	(flags) ← (status da comparação de AX com BX)
CMP BX, DX	(flags) ← (status da comparação de BX com DX)
	Demais combinações entre registradores de 16 bits (...X) são válidas

INSTRUÇÕES PARA COMPARAÇÕES ENTRE REGISTRADORES E DADOS DE 16 BITS

Mnemônico:	Descrição da instrução:
CMP AX, word	(flags) ← (status da comparação de AX com word)
CMP BX, word	(flags) ← (status da comparação de BX com word)
	Demais registradores de 16 bits (...X) são válidos

INSTRUÇÕES PARA DESVIO INCONDICIONAL

Mnemônico:	Descrição da instrução:
JMP endereço	IP ← endereço fornecido (salto dentro do segmento indicado por CS)

INSTRUÇÕES PARA DESVIO CONDICIONAL

Mnemônico:	Descrição da instrução:
JZ endereço	Se a flag zero indicar resultado igual a zero, IP ← endereço Se a flag zero indicar resultado diferente de zero, não há desvio
JNZ endereço	Se a flag zero indicar resultado igual a zero, não há desvio Se a flag zero indicar resultado diferente de zero, IP ← endereço
JC endereço	Se a flag carry indicar que houve um overflow, IP ← endereço Se a flag carry indicar que não houve um overflow, não há desvio
JNC endereço	Se a flag carry indicar que houve um overflow, não há desvio Se a flag carry indicar que não houve um overflow, IP ← endereço

INSTRUÇÕES PARA USO DE SUB-ROTINAS

Mnemônico:	Descrição da instrução:
CALL endereço	Chamada de subprotina IP ← endereço e Pilha Operacional ← (IP anterior + 2)
RET	Retorno de sub-rotina IP ← (dois bytes do topo da Pilha Operacional)

INSTRUÇÕES PARA USO DA PILHA OPERACIONAL

Mnemônico:	Descrição da instrução:
PUSH AX	Pilha Operacional ← AX
PUSH BX	Pilha Operacional ← BX
PUSH CX	Pilha Operacional ← CX
PUSH DX	Pilha Operacional ← DX
PUSHF	Pilha Operacional ← Registrador de flags
POP AX	AX ← Topo da Pilha Operacional
POP BX	BX ← Topo da Pilha Operacional
POP CX	CX ← Topo da Pilha Operacional
POP DX	DX ← Topo da Pilha Operacional
POPF	Registrador de flags ← Topo da Pilha Operacional

INSTRUÇÃO PARA REPETIÇÃO CÍCLICA

Mnemônico:	Descrição da instrução:
LOOP endereço	IP ← endereço (salto dentro do segmento indicado por CS) O desvio é realizado enquanto o conteúdo de CX for maior que zero Caso contrário a próxima instrução do programa é executada CX é o contador de repetições

INSTRUÇÃO SEM OPERAÇÃO

Mnemônico:	Descrição da instrução:
NOP	Não realiza operação alguma

INSTRUÇÃO DE PARADA

Mnemônico:	Descrição da instrução:
HLT	Produz a parada do processador, usada para a finalização de um processamento cíclico.

5.4 Bibliografia

Linguagem Assembly para IBM PC
Peter Norton e John Socha
Editora Campus

Guia do Programador para IBM PC
Peter Norton
Editora Campus

Desvendando o PC
Peter Norton
Editora Campus

PC Assembler usando DOS
Daniel G. A. Quadros
Editora Campus

Turbo Assembler e Macro Assembler – Guia do Usuário
Jeremias Pereira dos Santos
Mc Graw Hill

8086 e 8088 – Hardware, Software, Aplicações e Projetos
Wilson Alonso Dias Jr.
Mc Graw Hill

The 8088 Project Book
Robert Grossblatt
Tab Books Inc

The 8088 and 8086 Microprocessors
W. Triebel, A. Singh
Prentice-Hall International Editions

The 8086/8088 Family: Design, Programming, and Interfacing
John Uffenbeck
Prentice-Hall International Editions

PC: um Guia Prático de Hardware e Interfaceamento - 2ª Edição
Ricardo Zelenovsky e Alexandre Mendonça
MZ Editora Ltda.

Interrupções do MS-DOS - Guia de Consulta Rápida
Adlich
Novatec Editora

Notas de aulas diversas

2002/W.R.